2009

Dynamic Kernel Function for High-Speed Real-Time Fast Fourier Transform Processors

Yu-Heng George Lee
Wright State University

DYNAMIC KERNEL FUNCTION FOR HIGH-SPEED REAL-TIME FAST FOURIER TRANSFORM PROCESSORS

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

By

YU-HENG GEORGE LEE
B.S. Electrical Engineering, University of Wisconsin – Madison, 2002
M.S. Electrical Engineering, Wright State University, 2004

2009

Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

November 10, 2009

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Yu-Heng George Lee ENTITLED Dynamic Kernel Function for High-Speed Real-time Fast Fourier Transform Processors BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

_____
Chien-In Henry Chen, Ph.D.
Dissertation Director

_____
Ramana V. Grandhi, Ph.D.
Director, Ph.D. in Engineering Program

_____
Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Committee on
Final Examination

_____
Chien-In Henry Chen, Ph.D.

_____
Raymond E. Siferd, Ph.D.

_____
Zhiqiang John Wu, Ph.D.

_____
Bin Wang, Ph.D.

_____
Wen-Ben Jone, Ph.D.

ABSTRACT

Lee, Yu-Heng George. Ph.D. in Engineering Program, Department of Electrical Engineering, Wright State University, 2009. Dynamic Kernel Function for High-Speed Real-Time Fourier Transform Processors.

The fast Fourier transform (FFT) plays a critical role in many modern applications, such as acoustics, optics, telecommunications, wireless sensor networks, location sensing, patient monitoring, speech, signal detection, and image processing. The input dynamic range, data throughput rate, frequency resolution, bandwidth, design flexibility, hardware consumption, and power requirements for the various applications are vastly different, leading to significant research focusing on different aspects of FFT performance improvement.

The proposed dynamic kernel function uses an efficient fixed-point numerical representation of the twiddle factor and replaces the cumbersome multipliers with simple shift-and-add operations to enhance the data throughput rate for high-speed wideband signal detection. Numerical representation in hardware plays a role in determining the dynamic range and bit precision of FFT processors. Variable truncation scheme dynamically scales the computation data and maximizes the use of fixed-input and inter-stage wordlength in existing hardware efficient fixed-point FFT. The above data scaling algorithm enhances the dynamic range of fixed-point fixed-precision FFT designs and emulates the precision benefits of floating-point representation without complicated design additions. Novel algorithms and performance analysis for hardware efficient representation of twiddle factors are studied for multi-tone signal detection with dynamic kernel function FFT processors. The development of hardware performance estimation

models based on different number of bits used for dynamic kernel function shows the relative trade-off between kernel bits to FFT spurious-free dynamic range (SFDR) and phase performance.

A 2.048 GSPS fixed-point fixed-precision dynamic kernel function FFT processor with variable truncation scheme is proposed, developed and implemented for real-time wideband signal detection. Using an Atmel 10-bit ADC, two-tone real-time signal detection is demonstrated for a bandwidth of 912 MHz with 16 MHz channelization and output throughput rates of 62.5 ns. The proposed design has an averaged single signal SFDR of 26 dB and the ability to detect a weak input signal at -42 dBm. The overall dynamic range (DR) of the system is 45 dB. This is possible for a fixed-precision FFT design due to the embedded variable truncation scheme to extend the total DR while preserving the instantaneous dynamic range (IDR) relationship. Additional case studies utilizing different dynamic kernel function and inter-stage precision shows important area and performance trade-offs when utilized in a high-speed wideband FFT processor without the use of decimators.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I want to express my sincere gratitude to my advisor, Dr. Henry Chen, for his invaluable guidance, phenomenal patience, constant encouragement, and unconditional support. Dr. Chen is an extraordinary mentor and compassionate friend. He kindly challenged me on how to be a confident and effective instructor as well as a humble yet innovative researcher. His bold visions and unconventional directions expanded my perspectives on academia and life. I am forever indebted to his respected advices.

I would also like to recognize my supportive committee members. I am grateful to Dr. Raymond Siferd for giving me the opportunity to expand my horizons and the chance to further polish my research skills. I honor Dr. Zhiqiang Wu on invigorating my fervor on digital and wireless communications with his passionate teaching. I owe Dr. Bin Wang for serving as an excellent role model in research. And I appreciate Dr. Wen-Ben Jone for his advices on post graduate career options.

I value Dr. Kefu Xue on sharing his knowledgeable insights on digital signal processing while I performed my teaching assistant duties for his class. Special thanks to Dr. Marian Kazimierczuk for his supportive academic advices and encouraging remarks.

Furthermore, I acknowledge the Ph.D. in Engineering Program and Department of Electrical Engineering for their generous financial support to complete my studies. I am obliged to the dependable services of Alysoun Taylor, Sheila Hollenbaugh, Doug Supp, and Mike VanHorn. My special thanks to the current and former administrators in the Department of Electrical Engineering, especially, Dr. Kefu Xue, Dr. Fred Garber, Mr. Barry Woods, Vickie Slone, and Marie Donohue for their assistance.

I am delighted to be in the company of great colleagues and friends. I cherish the friendship of Sridhar Ramachandran, Mingzhen Wang, Kumar Yelamarthi, Saiyu Ren, Nisha Das, Andrew Kondrath, Harish Gopalakrishnan, Dakshina Murthy Bellur, Veda Prakash, Ethan Lin, Swaroop Guttenahalli, Rajasekhar Keerthi, James Helton, Vivek Sarathy, Ryan Bone, Dilip Srinivasa, Stephen Benson, and Arvind Vaidyanadeswaran.

Jocelyn, my honey, you grace my life with your presence. Thank you for your glamorous smiles and showering me with your love. I appreciate your understanding and complementary support during this critical moment of my journey.

Lastly, I would like to thank my family for their unrelenting love and unwavering support. My parents have made many sacrifices to help solidify my educational background. My father's consistent encouragements and delightful insights are essential in developing me into a more complete individual.

1. INTRODUCTION

Since the fabrication of the first integrated circuit (IC) on a semiconductor material in the late 1950s, the number of transistors that can be cost-effectively placed on a single IC has increased exponentially. This allows a very large scale integration (VLSI) of transistors, which just surpassed the one billionth mark in 2005 [1], to implement complex systems consisting of analog, digital, or mixed-signal designs for application specific integrated circuits (ASIC).

Computation intensive digital signal processing (DSP) algorithms such as the fast Fourier transform (FFT) has benefited from the advancement of fabrication technology. FFT processors can now be realized and real-time processed in reconfigurable IC devices, namely field programmable gate arrays (FPGA). The use of FPGA to develop prototype models of digital designs offers lower non-recurrent engineering (NRE) costs, reduced development time, easier debugging, and high design flexibility [2]. The push for digital receivers also reduced the NRE costs needed for tuning analog components.

1.1 Motivation

Recent consumer desires for high data transmission rates has prompted many multiple-input and multiple-output (MIMO) based designs for multiple carrier transmission over a wide frequency band, such as orthogonal frequency-division multiplexing (OFDM) modems. Advantages of such systems include higher spectral

efficiency and data throughput rate. The role of FPGA in digital wideband receiver and software defined radio design has recently experienced a tremendous growth. It provides engineers with a cost effective solution to easily adapt to changing standards and spectrum requirements when programmed with Xilinx System Generator for DSP or hardware description language (HDL). Software implementation of software defined radios also enables exciting fields such as cognitive radio. In addition, dedicated high speed multipliers on the FPGA provide an attractive platform for exploring DSP algorithms with high number of multiply and accumulate (MAC) operations.

Latest technological evolution in data conversion and FPGA allows the realization of high speed digital wideband receivers at speeds of 2-3 giga samples per second (GSPS) and boosts the study of real-time data acquisition systems. This figure of merit also prompts the status of FPGA as a suitable replacement for slower general purpose and digital signal processors.

This work is motivated by the design challenge to process the GHz sampled input without the use of decimators while maintaining a high output throughput rate for FFT prototyping. The lack of current study on the effect of kernel function bit representation to the overall FFT processor performance is also another stimulus of this work.

1.2 Problem Statement

In the field of psychology, perception is the process of attaining awareness or understanding sensory information, where information may come in the form radio frequency (RF) signal. When observing the frequency spectrum, a skilled human operator has an obvious advantage over a digital receiver in discriminating the incoming

information as signal or noise. The means to quantify the ability to distinguish between signal and noise are first attributed to the work of radar researchers in signal detection theory [3].

The accuracy and precision of signal determination is critical for radar systems to correctly identify the range, altitude, direction, or speed of stationary and moving objects such as mechanical vehicular systems, weather patterns, and terrain information. For a given bandwidth, the ability to correctly identify multi-tone signals with varying input amplitudes from noise are significant for post processing. However, the limitation of available hardware resources and bit-precision for a given system leads to functionality and performance trade-offs in wide bandwidth signal detection.

1.3 Dissertation Scope

The objective is to develop, implement, and evaluate a real-time digital microwave receiver system capable of identifying multi-tone signals with coarse frequency estimation. The system performance is constrained by limited hardware resource available with limited bit-precision. High speed analog-to-digital converters (ADC) and real-time processing of sampled time domain data are essential in satisfying the wide bandwidth and output throughput requirements. A high-speed fixed-precision fixed-point dynamic kernel function FFT with variable truncation is proposed, developed, and implemented for multiple signal determination in addition to weak signal detection. Novel algorithms and performance analysis for hardware efficient representation of twiddle factors are studied for multi-tone signal detection with fast Fourier transform

processors. The efficient use of the limited data precision through the system flow is also significant in the final digital microwave receiver performance.

In addition, implementing a fixed point fixed precision FFT presents a challenge to compete with the performance superiority of floating point FFT. The dynamic kernel function and variable truncation seeks a solution for minimizing hardware while not sacrificing the FFT performance with real-time processing.

1.4 Overview

Chapter 2 begins with a general overview of various algorithms to efficiently compute the discrete Fourier transform (DFT). Chapter 3 provides literature review of recent fast Fourier transform (FFT) related research areas. The core research work begins with Chapter 4's introduction to dynamic kernel function and related performance trade-offs with various word lengths between FFT stages. The overall digital microwave receiver system and implementation is presented in Chapter 5. The tools and design flow demonstrated in Chapter 6 will supply future researchers with more efficient prototype design cycles for digital wideband receiver implementations. The performance analysis of digital receiver implemented on the FPGA is discussed in Chapter 7. The final chapter summarizes the overall research work with future directions for this study.

2. DISCRETE FOURIER TRANSFORM ALGORITHM

Fast Fourier transform is an efficient algorithm to calculate the discrete Fourier transform. It transforms a set of sampled time domain data over a finite interval of time, into a discrete set of coarse frequencies, represented by the frequency bins. FFT acts as a bank of narrow-band filters followed by a corresponding set of detectors. The detectors are indicators for the total energy passed by each filter. However, these filters are not very frequency selective, requiring the need for window functions to weight the sampled time domain data.

Many digital signal processing and linear systems texts provide a good reference on the subject of FFT and may be consulted for more detailed derivations for efficient methods of computing the DFT [4-10]. This chapter begins with a brief history on the development of efficient algorithm for the DFT. The text is followed by some discussion and derivations for the commonly used FFT algorithms. The discussion of FFT algorithms is coarsely classified and sub-divided as common-factor, prime-factor, and others.

2.1 Historical Perspective

In Cooley and Tukey's (1965) seminal work on efficiently computing the discrete Fourier transform [11], they exploited the symmetry and periodicity of the DFT kernel function $e^{j\theta}$ to simplify the number of operations needed to calculate a DFT. Symmetry

refers to the even and odd symmetries of the cosine and sine terms of the complex phasor $e^{j\theta}$ expression in rectangular form using Euler's identity. The Tukey and Cooley algorithm uses a divide and conquer approach to compute the DFT with less mathematical operations using a recursive composite of progressively shorter length DFTs. The divide and conquer approach can be further categorized into common-factor and prime-factor algorithms, which determines the type of lengths required of the shorter DFTs.

Heideman [12] traced the origins of FFT dating back to Gauss in 1805, two years before Fourier's landmark work on the Fourier series and transforms. The ideas of exploiting symmetry and periodicity to compute the DFT were first proposed in separate papers written by Runge in 1903 and 1905 [13, 14]. Earlier works of Danielson and Lanczos in 1942 [15] are referenced by Tukey and Cooley for describing a type of FFT algorithm used X-ray scattering experiments with similar computational complexity of close to N log N.

Prior to Cooley and Tukey's work, much of the work with discrete Fourier transform used small data sequences to make hand calculations possible. Advances in semiconductor technology in the late 50s provided a new gateway for computing discrete Fourier transforms using digital integrated circuits, which prompts research into highly computationally efficient algorithms, known as the fast Fourier transform (FFT), to compute DFT lengths not feasible for hand calculations. In 1967 and 1969, Singleton demonstrated the use of DFT in convolution, correlation, and spectrum analysis [16] as well as introducing the mixed radix FFT in [17]. This spurred further development of FFT algorithms.

Duhamel and Hollmann introduced split-radix FFT (SRFFT) algorithm in 1984 and 1986 [18,19]. Swartztrauber exploited the data symmetrical properties to reduce the computation time [20].

Alternate forms of DFT computation include the rearrangement of DFT to compute it as a linear convolution. Goertzel indicated that the DFT can be computed using linear filtering in 1968 with modest observed computational savings [21]. Bluestein formulated the DFT as a chirp linear filtering operation in 1968 [22], which led to the development of the chip-z transform by Rabiner in 1969 [23].

Apart from the above mentioned methods of using common-factors or linear convolution to compute the DFT, the prime-factor based FFTs also received some attention. These include works by Rader in 1968 [24], Good in 1971 [25], and the Winograd algorithm in 1978 [26].

2.2 Tukey and Cooley Fast Fourier Transform

Fast Fourier transform is an efficient method of computing the DFT, not an approximate solution. Given an input data sequence *x(n)* of length *N*, the DFT of *x(n)*, defined as *X(k)*, is expressed as,

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N} \quad , n = 0, 1, \ldots, N-1, k = 0, 1, \ldots, N-1 \qquad (2.2.1)$$

where *n* is the time index, *k* is the frequency index, and *N* is the total number of samples. Using the more compact notation for the kernel function in Eq. (2.2.2),

$$W_N^{nk} = e^{-j2\pi nk/N} \qquad (2.2.2)$$

Eq. (2.2.1) may be re-expressed as,

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk} \text{ , } (n, k) = 0, 1, \ldots, N - 1 \tag{2.2.3}$$

The kernel function, also known as the twiddle or phase factor, plays an important role in the reduction of computational complexity. The gist of the FFT algorithms utilizes the symmetric and periodic properties of the kernel functions to reduce the computational complexity of the DFT. Using Euler's formula in Eq. (2.2.4), the above mentioned properties can be closely studied.

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{2.2.4}$$

Eq. (2.2.2) may also be expressed in rectangular form utilizing Euler's formula.

$$W_N^{nk} = e^{j2\pi nk/N} = \cos\left(\frac{2\pi nk}{N}\right) + j\sin\left(\frac{2\pi nk}{N}\right) \tag{2.2.5}$$

To exploit the symmetric property, the *k* in Eq. (2.2.5) is replaced by *k+N/2*.

$$\begin{aligned} W_N^{n(k+N/2)} &= e^{-j2\pi n(k+N/2)/N} \\ &= e^{-j2\pi nk/N} e^{-j2\pi n(N/2)/N} \\ &= e^{-j2\pi nk/N} e^{-j\pi n} \end{aligned} \tag{2.2.6}$$

Utilizing Euler's formula,

$$e^{-j\pi n} = \cos(-\pi n) + j\sin(-\pi n) = -1 + j(0) = -1 \tag{2.2.7}$$

combining Eqs. (2.6) and (2.7),

$$\begin{aligned} W_N^{n(k+N/2)} &= e^{-j2\pi nk/N} e^{-j\pi n} \\ &= -e^{-j2\pi nk/N} = -W_N^{nk} \end{aligned} \tag{2.2.8}$$

the symmetric property is exploited. This indicates that half of the kernel functions may be replaced by using its complex conjugate.

The periodic property of the kernel function is proved by substituting the k in Eq. (2.2.5) is with $k+N$.

$$\begin{aligned} W_N^{n(k+N)} &= e^{-j2\pi n(k+N)/N} \\ &= e^{-j2\pi nk/N} e^{-j2\pi nN/N} \\ &= e^{-j2\pi nk/N} e^{-j2\pi n} \end{aligned} \tag{2.2.9}$$

With the aid of the Euler formula,

$$e^{-j2\pi n} = \cos(-2\pi n) + j\sin(-2\pi n) = 1 + j(0) = 1 \tag{2.2.10}$$

the periodic relationship is established by combining Eqs. (2.2.9) and (2.2.10).

$$\begin{aligned} W_N^{n(k+N)} &= e^{-j2\pi nk/N} e^{-j2\pi n} \\ &= e^{-j2\pi nk/N} = W_N^{nk} \end{aligned} \tag{2.2.11}$$

Cooley and Tukey's FFT algorithm utilizes the above properties with the divide and conquer approach. The N-point DFT is first decomposed into two shorter length DFTs. N is first factored as a product of two integers,

$$N = N_1 N_2 \tag{2.2.11}$$

where N_1 and N_2 may be any integer number, not necessarily a prime number. The input data sequence x(n) is then mapped into a two dimensional array $x(n_1, n_2)$, where n_1 may be any integer value between 0 and (N_1-1), and n_2 may be any integer value between 0 and (N_2-1). The input data sequence number n in Eq. (2.2.3) can be substituted by the following.

$$n = n_1 + n_2 N_1 \tag{2.2.12}$$

Depending on the type of algorithm used, there are different input and output data mapping schemes. N_1 and N_2 point DFTs are performed with kernel function multiplication between the two DFT operations. Another set of mapping is required for

storing the computed DFT output values *X(k)*, and indices p and q are introduced to map the output as *X(p,q)*. Similar to n_1 and n_2, p may be any integer value between 0 and (N_1-1), and q may be any integer value between 0 and (N_2-1). The output data sequence index k in Eq. (2.2.3) can be substituted by the following.

$$k = N_2 p + q \tag{2.2.13}$$

The final results are re-ordered using the above indexing scheme and read.

Using the above algorithm, Eq. (2.2.3) can be manipulated with Eqs. (2.2.12) and (2.2.13).

$$X(p,q) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(n_1,n_2) \cdot W_N^{(n_1+n_2N_1)(N_2p+q)} \tag{2.2.14}$$

And

$$W_N^{(n_1+n_2N_1)(N_2p+q)} = W_N^{n_1N_2p} \cdot W_N^{n_1q} \cdot W_N^{n_2N_1N_2p} \cdot W_N^{n_2N_1q} \tag{2.2.15}$$

However, Eq. (2.2.15) may be further reduced to,

$$W_N^{n_1N_2p} = W_{N/N_2}^{n_1p} = W_{N_1}^{n_1p} \tag{2.2.16}$$

$$W_N^{n_2N_1N_2p} = W_N^{n_2Np} = 1 \tag{2.2.17}$$

$$W_N^{n_2N_1q} = W_{N/N_1}^{n_2q} = W_{N_2}^{n_2q} \tag{2.2.18}$$

Using the relationships found in Eqs. (2.2.16), (2.2.17), and (2.2.18), Eq. (2.2.14) can be expressed as

$$X(p,q) = \sum_{n_2=0}^{N_2-1} \left\{ W_N^{n_1q} \left[\sum_{n_1=0}^{N_1-1} x(n_1,n_2) \cdot W_{N_1}^{n_1p} \right] \right\} W_{N_2}^{n_2q} \tag{2.2.19}$$

The divide and conquer algorithm can be summed up in the following 5 steps.

1. Store the signal column-wise, using the index mapping,

$$n = n_1 + n_2 N_1 \tag{2.2.20}$$

2. Compute the N_1-point DFT, $F(n_1,q)$ of each row.

$$F(n_1, q) = \sum_{n_1=0}^{N_1-1} x(n_1, n_2) \cdot W_{N_1}^{n_1 p} \tag{2.2.21}$$

3. Multiply the resulting array by the phase factors $W_N^{(n1)q}$, using indices n_1 and q.

$$G(n_1, q) = F(n_1, q) \cdot W_N^{n_1 q} \tag{2.2.22}$$

4. Compute the N_2-point DFT of each column.

$$X(p, q) = \sum_{n_2=0}^{N_2-1} G(n_1, q) \cdot W_{N_2}^{n_2 q} \tag{2.2.23}$$

5. Read the resulting array row-wise, using the index mapping,

$$k = N_2 p + q \tag{2.2.24}$$

The algorithm can also be depicted graphically, using an example of 15-point FFT, with N1 = 3 and N2 = 5, Fig. 2.1 explains the methodology used pictorially.

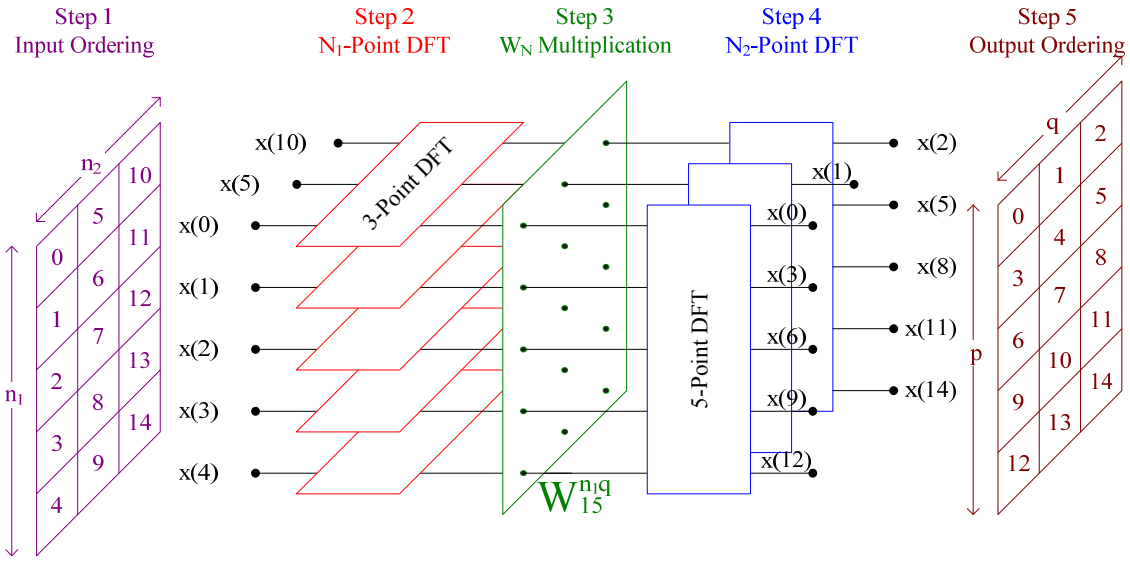

Fig. 2.1 Decomposing 15-point FFT into 3-point and 5-point DFTs

Both the common-factor and prime-factor FFTs start with this algorithm with varying procedures within the five steps described above. The type of changes ranges from further decomposition of the N-point DFT into lengths of r, (radix-2, radix-4, radix-r), to modifications of input and output mappings [27] (decimation-in-time, decimation-in-frequency, prime-factor). Additional studies have incorporated different radix-r DFTs within the decomposed DFTs (mixed-radix, split-radix). These variations were mostly derived from Cooley and Tukey's algorithm, which further emphasized the significance of their work.

2.3 Radix-Based FFT Algorithms

Radix-r based FFT is a class of common-factor FFT, where the length of the DFT, N, is decomposed into k stages of recursively decimated DFTs with the common factor r ($N= r^k$) to achieve the computation advantage of $O(N \log_r N)$. The word radix is used to describe the common factor used. The input and output data mappings are based on Cooley and Tukey's methodology, where the N-point sequence is mapped into a *r* by *(N/r)* array. The most widely used factors for *r* is based on powers of two, specifically radix-2 and radix-4. Radix-2 and radix-4 FFTs are commonly adopted for implementation due to the regularity of the butterfly structure and design simplicity. Depending on the input and output ordering, radix-2 FFTs can be further classified as decimation-in-time (DIT) and decimation-in-frequency (DIF).

2.3.1 Radix-2 Decimation-in-Time

Following the divide-and-conquer algorithm, the N-point sequence of the FFT is decomposed into lengths of $N_1 = 2$ and $N_2 = N/2$. Starting from the DFT expression,

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk} \tag{2.3.1}$$

The N-point data sequence is divided into two N/2-point data sequences,

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) \cdot W_N^{nk} \\ &= \sum_{n=0}^{(N/2)-1} x(2n) \cdot W_N^{2nk} + \sum_{n=0}^{(N/2)-1} x(2n+1) \cdot W_N^{(2n+1)k} \\ &= \sum_{n=0}^{(N/2)-1} x(2n) \cdot W_{N/2}^{nk} + W_N^k \cdot \sum_{n=0}^{(N/2)-1} x(2n+1) \cdot W_{N/2}^{nk} \\ &= X_{even}(k) + W_N^k \cdot X_{odd}(k) \end{aligned} \tag{2.3.2}$$

where

$$X_{even}(k) = \sum_{n=0}^{(N/2)-1} x(2n) \cdot W_{N/2}^{nk} \text{ , } k = 0, 1, \ldots, N - 1 \tag{2.3.3}$$

and

$$X_{odd}(k) = \sum_{n=0}^{(N/2)-1} x(2n+1) \cdot W_{N/2}^{nk} \text{ , } k = 0, 1, \ldots, N - 1 \tag{2.3.4}$$

x(2n) and x(2n+1) corresponds to even and odd numbered samples of the input x(n). Eq. (2.3.2) corresponds to the first step in the DIT algorithm and the computation is illustrated in Fig. 2.2 for an 8-point FFT.

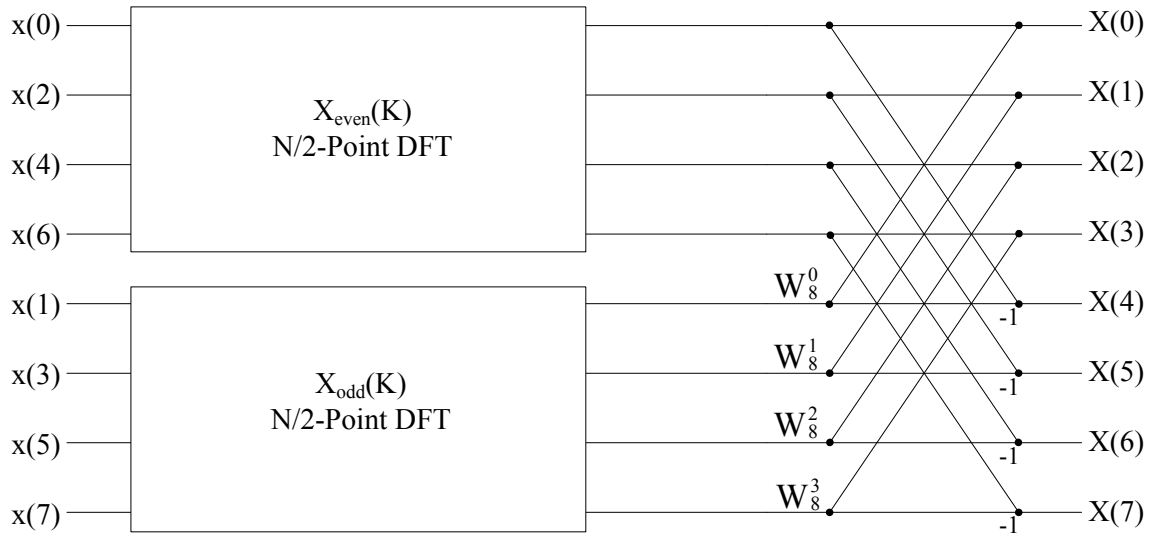

Fig. 2.2 First Step in the decimation-in-time for 8-point FFT

Eqs. (2.3.3) and (2.3.4) may be divided further into N/4-point DFT sections, eventually ending up with 2-point DFTs for N=8, as shown in Fig. 2.3.

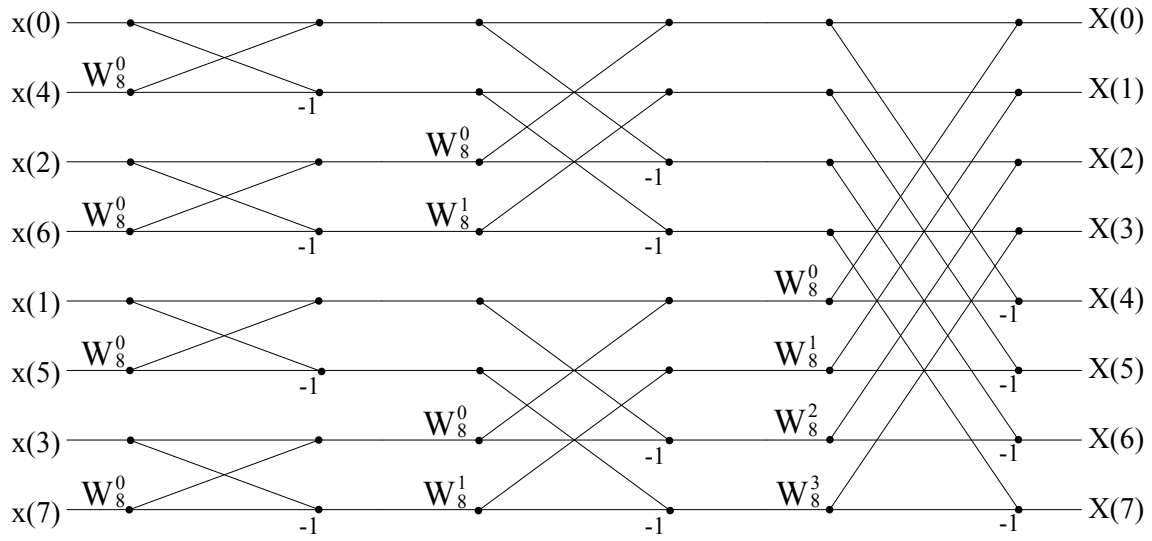

Fig. 2.3 8-point decimation-in-time FFT

In Fig. 2.3, the time domain input samples *x(n)* were recursively decimated into even and odd sequences, justifying with the naming convention used. Fig. 2.4 shows the radix-2 DIT butterfly,

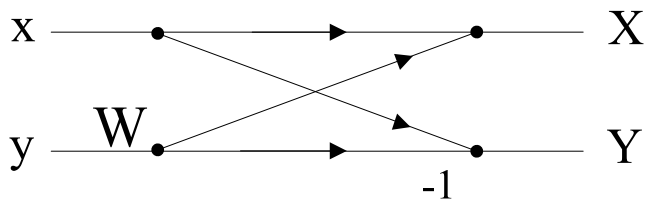

Fig. 2.4 Radix-2 DIT butterfly

where x and y are inputs, X and Y are outputs, and W is a predetermined kernel function, yielding,

$$X = x + W \cdot y \tag{2.3.5}$$

$$Y = x - W \cdot y \tag{2.3.6}$$

Additional advantage is observed from Eqs. (2.3.5) and (2.3.6), the term $W \cdot y$ will only need to be calculated once in the hardware implementation.

2.3.2 Radix-2 Decimation-in-Frequency

Similar to decimation-in-time, the N-point sequence of the decimation-in-frequency FFT is decomposed into lengths of $N_1 = N/2$ and $N_2 = 2$. The N-point data sequence is divided into two N/2-point parts in the frequency domain,

$$\begin{aligned} X(2k) &= \sum_{n=0}^{N-1} x(n) \cdot W_N^{n2k} = \sum_{n=0}^{N-1} x(n) \cdot W_{N/2}^{nk} \\ &= \sum_{n=0}^{(N/2)-1} x(n) \cdot W_{N/2}^{nk} + \sum_{n=(N/2)}^{N} x(n) \cdot W_{N/2}^{nk} \\ &= \sum_{n=0}^{(N/2)-1} x(n) \cdot W_{N/2}^{nk} + \sum_{n=0}^{(N/2)-1} x(n+N/2) \cdot W_{N/2}^{(n+N/2)k} \end{aligned} \tag{2.3.7}$$

where,

$$W_{N/2}^{(n+N/2)k} = W_{N/2}^{nk} \cdot W_{N/2}^{(N/2)k} = W_{N/2}^{nk} \cdot 1 = W_{N/2}^{nk} \tag{2.3.8}$$

Thus,

$$\begin{aligned} X(2k) &= \sum_{n=0}^{(N/2)-1} x(n) \cdot W_{N/2}^{nk} + \sum_{n=0}^{(N/2)-1} x(n+N/2) \cdot W_{N/2}^{(n+N/2)k} \\ &= \sum_{n=0}^{(N/2)-1} x(n) \cdot W_{N/2}^{nk} + \sum_{n=0}^{(N/2)-1} x(n+N/2) \cdot W_{N/2}^{nk} \\ &= \sum_{n=0}^{(N/2)-1} \left[x(n) + x(n+N/2) \right] \cdot W_{N/2}^{nk} \end{aligned} \tag{2.3.9}$$

And for the second half,

$$\begin{aligned} X(2k+1) &= \sum_{n=0}^{N-1} x(n) \cdot W_N^{n(2k+1)} = \sum_{n=0}^{N-1} x(n) \cdot W_N^{n2k} \cdot W_N^n = \sum_{n=0}^{N-1} x(n) \cdot W_{N/2}^{nk} \cdot W_N^n \\ &= \sum_{n=0}^{(N/2)-1} x(n) \cdot W_{N/2}^{nk} \cdot W_N^n + \sum_{n=(N/2)}^{N} x(n) \cdot W_{N/2}^{nk} \cdot W_N^n \\ &= \sum_{n=0}^{(N/2)-1} x(n) \cdot W_{N/2}^{nk} \cdot W_N^n + \sum_{n=0}^{(N/2)-1} x(n+N/2) \cdot W_{N/2}^{(n+N/2)k} \cdot W_N^{(n+N/2)} \\ &= \sum_{n=0}^{(N/2)-1} x(n) \cdot W_{N/2}^{nk} \cdot W_N^n + \sum_{n=0}^{(N/2)-1} x(n+N/2) \cdot W_{N/2}^{nk} \cdot W_N^{(n+N/2)} \end{aligned} \tag{2.3.10}$$

where,

$$\begin{aligned} W_N^{(n+N/2)} &= W_N^n \cdot W_N^{(N/2)} = W_N^n \cdot W_{2N}^N \\ &= W_N^n \cdot e^{-j2\pi N/2N} = W_N^n \cdot e^{-j\pi} = W_N^n \cdot (-1) \end{aligned} \tag{2.3.11}$$

Thus,

$$
\begin{aligned}
X(2k+1) &= \sum_{n=0}^{(N/2)-1} x(n)\cdot W_{N/2}^{nk}\cdot W_N^n + \sum_{n=0}^{(N/2)-1} x(n+N/2)\cdot W_{N/2}^{nk}\cdot W_N^{(n+N/2)} \\
&= \sum_{n=0}^{(N/2)-1} x(n)\cdot W_{N/2}^{nk}\cdot W_N^n + \sum_{n=0}^{(N/2)-1} x(n+N/2)\cdot W_{N/2}^{nk}\cdot W_N^n\cdot(-1) \\
&= \sum_{n=0}^{(N/2)-1} x(n)\cdot W_{N/2}^{nk}\cdot W_N^n - \sum_{n=0}^{(N/2)-1} x(n+N/2)\cdot W_{N/2}^{nk}\cdot W_N^n \\
&= \sum_{n=0}^{(N/2)-1} \left\{\left[x(n)-x(n+N/2)\right]\cdot W_N^n\right\}\cdot W_{N/2}^{nk}
\end{aligned}
\tag{2.3.12}
$$

The final expressions for Eqs. (2.3.9) and (2.3.12) are,

$$
X(2k) = \sum_{n=0}^{(N/2)-1}\left[x(n)+x(n+N/2)\right]\cdot W_{N/2}^{nk} \tag{2.3.13}
$$

$$
X(2k+1) = \sum_{n=0}^{(N/2)-1}\left\{\left[x(n)-x(n+N/2)\right]\cdot W_N^n\right\}\cdot W_{N/2}^{nk} \tag{2.3.14}
$$

Eqs. (2.3.13) and (2.3.14) corresponds to the first step in the DIF algorithm and the computation is illustrated in Fig. 2.5 for an 8-point FFT.

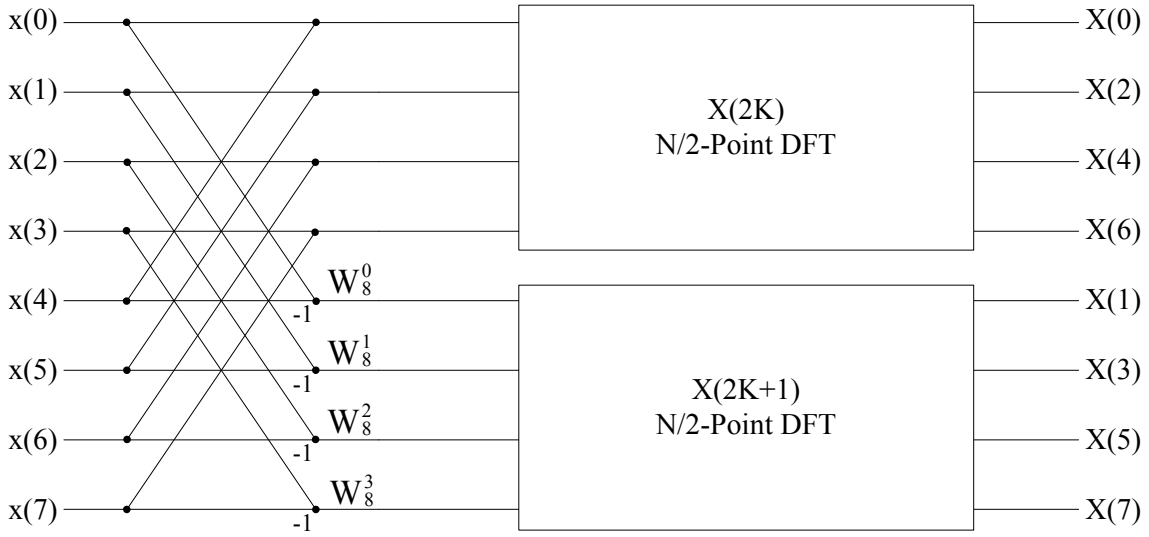

Fig. 2.5 First Step in the decimation-in-frequency for 8-point FFT

Eqs. (2.3.13) and (2.3.14) may be divided further into N/4-point DFT sections, eventually ending up with 2-point DFTs for N=8, as shown in Fig. 2.6.

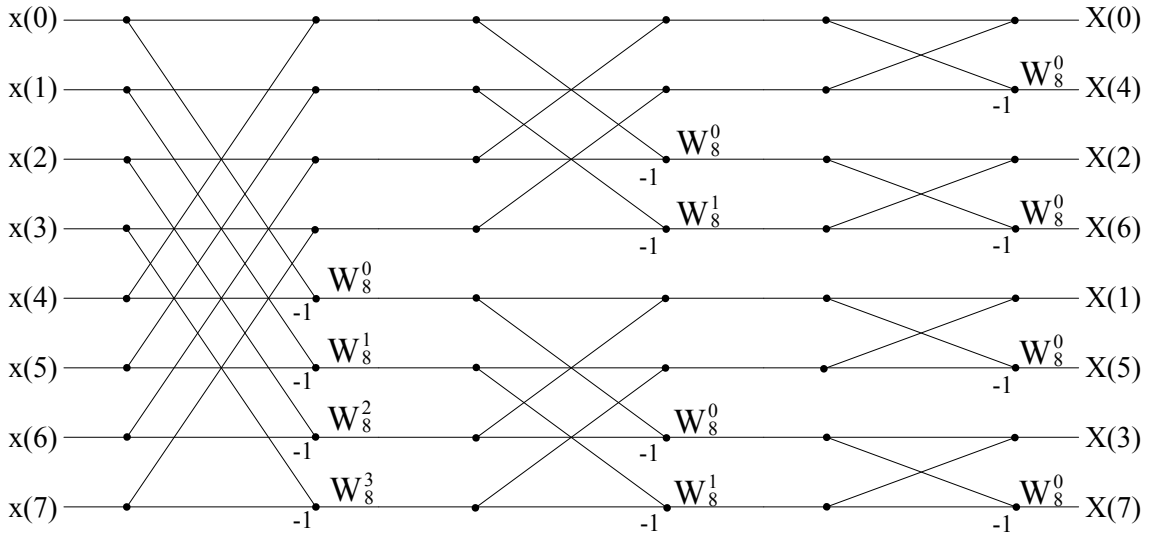

Fig. 2.6 8-point decimation-in-frequency FFT

In Fig. 2.6, frequency domain output samples *X(k)* were recursively decimated into even and odd sequences. Fig. 2.7 shows the radix-2 DIT butterfly,

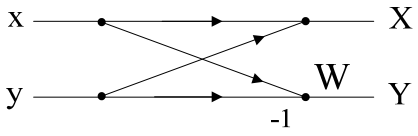

Fig. 2.7 Radix-2 DIF butterfly

where x and y are inputs, X and Y are outputs, and W is a predetermined kernel function, yielding,

$$X = x + y \tag{2.3.15}$$

$$Y = (x - y) \cdot W \tag{2.3.16}$$

In terms of hardware usage, both DIT and DIT uses the same number of multiplier and adders. However, when the kernel function W is approximated in the hardware implementation, DIF may have a slight advantage since half of the points in each stage will not be affect by the corresponding arithmetic error.

2.3.3 Radix-4 and Higher Radices

The derivation for the radix-4 algorithm is similar to the methodology used above. The radix-4 butterfly is shown below.

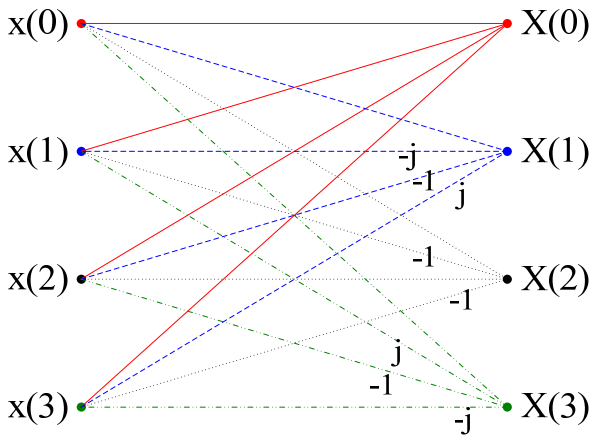

Fig. 2.8 Radix-4 butterfly

where the outputs are,

$$X(0) = x(0) + x(1) + x(2) + x(3) \tag{2.3.17}$$

$$X(1) = x(0) - j \cdot x(1) - x(2) + j \cdot x(3) \tag{2.3.18}$$

$$X(2) = x(0) - x(1) + x(2) - x(3) \tag{2.3.19}$$

$$X(3) = x(0) + j \cdot x(1) - x(2) - j \cdot x(3) \tag{2.3.20}$$

One radix-4 butterfly performs the work of four radix-2 butterflies and saves about 25% of complex multipliers. However, a full-custom design of high-point FFT

based on the radix-4 algorithm will be more complicated and requires more debugging time.

Higher radix FFT algorithms are uncommon because they involve more complex computation and dataflow schemes. The number of complex multipliers will also decrease proportionally with higher radix. Unfortunately, similar to the radix-4 algorithm, a full custom design will take longer to design and optimize for high throughput systems.

2.3.4 Mixed-Radix and Split-Radix

These two types of algorithms are often confused with one another. The mixed-radix FFT decomposes DFT into several different shorter DFT with relatively prime lengths, whereas the split-radix algorithm improves upon the *X(2k+1)* term in radix-2 DIF butterfly. The odd numbered samples *X(2k+1)* requires a pre-multiplication of the input sequence with the kernel function W_N^n. A radix-4 decomposition for *X(2k+1)* radix-2 DIF samples produces some computational efficiency with fewer multiplications when decomposed into *X(4k+1)* and *X(4k+3)*. The basic split-radix butterfly is shown in Fig. 2.9.

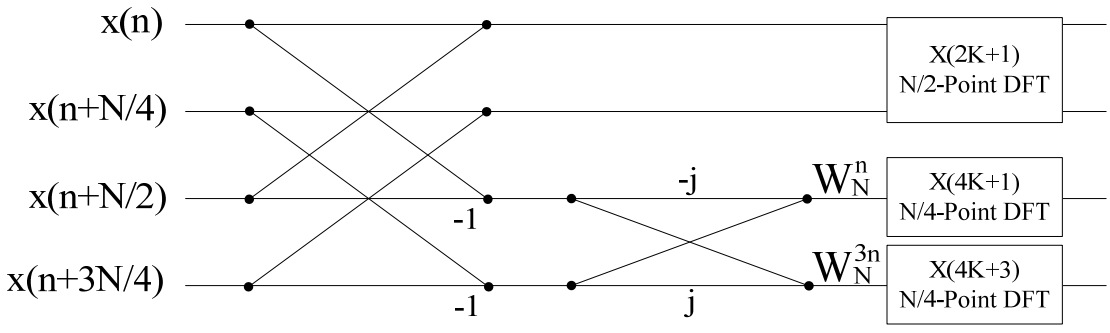

Fig. 2.9 Split-radix butterfly

2.3.5 Relevant Computational Efficiency

While Duhamel proposed the split-radix algorithm, a comparison of the number of adders and multipliers are also included, which provides a good insight into the amount of hardware efficiencies required for each FFT implementation.

Table 2.1 Number of Non-Trivial Multiplication and Additions for N-Point FFT [16]

	Real Multiplication				Real Additions			
N	Radix-2	Radix-4	Radix-8	Split-Radix	Radix-2	Radix-4	Radix-8	Split-radix
16	24	20		20	152	148		148
32	88			68	408			388
64	264	208	204	196	1032	976	972	964
128	712			516	2054			2308
256	1800	1392		1284	5896	5488		5380
512	4360		3204	3076	13566		12420	12292
1024	10248	7856		7172	30728	28336		27652

2.4 Other FFT Algorithms

Several other types of FFT algorithm will be briefly reviewed in this section for alternative methods of computing DFT.

Good Algorithm (Prime-factor FFT) – This algorithm uses a similar divide-and-conquer approach, where the decomposed DFT lengths are relatively prime without common factors between the smaller DFT lengths. The indices needed for the input and output mappings are complex and required the use of the Chinese Remainder Theorem (CRT). Aside from the above mentioned constraints, the methodology still follows the Cooley and Tukey algorithm.

Goertzel Algorithm – Goertzel exploited the periodicity of phase factors and allows the computation of DFT as a linear filtering operation. The DFT is calculated by passing blocks of input data into parallel bank of N single-pole filters. Each filter has a pole at the

corresponding frequency bins of DFT. However, the computation complexity of this algorithm does not vary much from the direct computation of the DFT.

Winograd Algorithm – This is a type of prime-factor FFT where the decomposed blocks of DFT use a highly efficient convolution. For a N-point Winograd FFT, the computation complexity is only *O(N)*, much more efficient than the common-factor FFTs. Unfortunately, the trade-off is the design structure of this FFT is complex and irregular with higher number of additions required.

3. PRECISION, ARCHITECTURE, AND POWER OF FFT PROCESSORS

The discrete Fourier transform is widely used in many science and engineering disciplines due to its computational efficiency [4]. It plays a critical role in many modern applications, such as acoustics, optics, telecommunications, wireless sensor networks, location sensing, patient monitoring, speech, signal, and image processing. [11,28,29,30].

The input dynamic range, data throughput rate, frequency resolution, bandwidth, design flexibility, hardware consumption, and power requirements for the various applications are vastly different, leading to many significant researches focusing on different aspects of FFT performance improvement. This chapter highlights the precision, architecture, butterfly structure, low-power design, and digital receiver studies to accommodate the above design specifications. Commonly used performance evaluation metrics are discussed at the end of the Chapter as well.

3.1 Numerical Representation

The numerical representation in hardware design is an important factor to consider prior to any hardware design. The number of bits used for the wordlength will ultimately determine input dynamic range, data throughput rate, and hardware consumption of the FFT processor. An enhanced signal-to-noise ratio performance is also demonstrated for increased wordlength in previous works [29,31].

Typically, the designer will need to choose from fixed-point, floating-point, or block floating-point notations to represent the arithmetic computation results. Fixed-point numbers uses a fixed number of bits to describe a fractional notation, either integer or fractional value by keeping track of the binary point. In contrast, floating-point processors use a mantissa and exponent to characterize a fixed-point number and the number of places the binary point must be shifted by. The number of bits used for the fixed-point number or mantissa will determine the FFT's dynamic range and the bit precision of signal representation.

The block floating point concept uses the floating-point representation for the numerical outputs of each FFT stage while sharing the exponent component amongst all mantissas. This leads to higher dynamic range while lower the hardware requirement needed with the floating point numbers [32].

Many dynamic data-scaling algorithms based on the concept of block floating point were proposed to reduce the inter-stage wordlength of the FFT [33-37]. However, many of these are based on a feedback loop model, where the previous exponent is used to evaluate against the incoming signals.

Variable truncation scheme (VTS) is proposed in this research work to provide a more convenient method of dynamically scaling the input data and inter-stage wordlength precision. As the sampled input data progresses through the FFT stages, the arithmetic results will progressive grow larger based on the current set of sampled input data. The bit growth for each stage is limited to 1-bit, thus VTS will only need to make the decision based on a 1-bit difference. The decisions made for scaling or not scaling are directly passed to the output of the FFT, where the final computed frequency components will all

be scaled accordingly. In relation with block floating-point implementation, the exponent component of our design is only 1 bit for every stage.

3.2 Architecture

During this phase of the design process, the wordlength of the FFT is already defined. The architecture determines the amount of hardware resource consumed, the expected output data throughput rate, and the power associated with the design.

The five most recently explored FFT architectures are single memory [38], dual-memory [39], pipelined [30], array [40], and cached-memory architecture [41]. The single memory architecture uses the DFT butterfly to process and store the computed data in the same memory location, yield the smallest hardware cost. The dual-memory uses two memory bank locations to allow the data to continually flow through the DFT butterfly from one memory and store the results in the other. The process is repeated alternatively. The cached-memory architecture inserts a cache between the processing and memory blocks in the single memory architecture. This approach results in a low-power resource efficient design, but additional functional unit insertions are needed in addition to increased controller complexity. The array architecture splits the FFT into separate independent single processor cores with local memory buffers to enhance the data processing rate. The data throughput rate is very high, but the power consumption is the highest among the discussed architectures.

For high throughput systems, pipelined architecture is often used. There are two sub-classes under this category. For a N-point FFT without folding, memory elements are inserted between each stage of the FFT. This is classified as a multipath delay

commutator (MDC) where the data processing is in parallel. When the butterfly hardware within each stage is reused to process all the data in the same stage, the datapath becomes linear, yielding the single-path delay commutator. A mix of single-path and multi-path delay commutator can be applied to the FFT to maintain the throughput requirement without increasing the hardware resource required. The dynamic kernel function FFT implemented in this dissertation follows the pipelined architecture.

For defining the architectural efficiency of the FFT, often the number of operations per second is observed and compared against other works.

3.3 Butterfly Implementation and Power

Memory access in low-power FFT design dominates the power consumption in FFT processors [42]. Memory access to acquire the kernel function stored can be further reduced with the proper coefficient ordering [43]. In addition, reducing the switching activity of multipliers has shown to decrease power consumption in FIR filters and equalizers [44]. Various attempts are made by replacing the twiddle factor multiplication with shift-and-add operation [45] and simplifying the twiddle factor representation with fixed integer constants [46] for power reduction.

The dynamic kernel function efficiently maps the twiddle factor into a n-bit integer format and replaced the multiplication with shift-and-add operation to reduce design complexity associated with each FFT butterfly. This shows good potential for future low-power applications while the reduced complexity improves the data throughput rate of the system.

3.4 Digital Microwave Receivers

The design of multi-tone wideband receiver for the detection of weak signals having unknown carrier frequency is a challenge due to the presence of thermal noise in the frequency spectrum as well as the occurrence of signal side-lobes and spurs generated by the receiver. The channelized receiver based on fast Fourier transform (FFT) digital filter banks is an important category of wideband digital receiver design [47]. The advantage of this receiver is the ability to detect multiple signals in contrast with conventional instantaneous frequency measurement (IFM) receiver which can only detect one signal. The reduction of FFT complexity is crucial to minimize the amount of hardware and power consumption.

Previous approaches to FFT complexity reduction include the monobit receiver, which accepts a 1-bit analog to digital converter (ADC) output to eliminate or minimize the need for multiplication. The monobit wideband receiver in [48] can detect and process two signals and achieve a two-tone signal instantaneous dynamic range (IDR) of 5 dB with a 1 GHz bandwidth. The two-tone signal IDR is the power ratio between the strong and weak signals. This monobit receiver included a 2-bit ADC and used a 4-point kernel function 256-point FFT.

The enhancements of the monobit wideband digital receiver presented in [49] replaced the multiplication function with shift and add operations in addition to the use of a 12-point kernel function. This receiver had a two-tone signal IDR of 18 dB with a second signal false alarm rate of less than 1 %. The improved receiver included 4 bit ADC, 12-point kernel function 256-point FFT and a super resolution block. The super resolution block implemented a compensation technique to improve the two-tone signal

IDR. The augmentation of Kaiser window function to this receiver in [50] increased the two-tone signal IDR to 24 dB with 80 % second signal detection rate.

The analysis in [51] used 4, 12, 16, 24, and 32-point fixed kernel functions to implement the 256-point FFT. This yielded single signal spurious free dynamic ranges (SFDR) of 4.98, 13.63, 21.30, 21.50, and 24.74 dB with the same measures for the single signal dynamic range (DR). The two-tone signal IDR is reported as 3.74, 11.58, 18.94, 19.02, and 21.80 dB with 90 % signal detection rate.

3.5 Performance Metrics

For digital receivers, the following figures of merits are typically used for performance evaluation.

Bandwidth – The difference between the detectable lower and upper frequency in a signal spectrum.

Frequency Resolution – The minimum frequency difference between two resolvable input sinusoids in the frequency spectrum.

Dynamic Range – The ratio between highest and lowest time-domain input amplitudes acceptable by the receiver. The ideal dynamic range of the receiver using a n-bit ADC is about 6*n dB [52].

Spurious-Free Dynamic Range (SFDR) – The ratio between the detected signal versus the highest noise spur, which includes harmonics.

Instantaneous Dynamic Range (IDR) – The time-domain power ratio between the strong and weakest detectable two-tone signals.

Detection Rate – The percentage of detected signals in a given time frame.

False Alarm Rate – The probability of false detection.

4. DYNAMIC KERNEL FUNCTION FFT ALGORITHM

The challenge for a hardware implementation of FFT, fixed or floating-point, is to find an accurate and cost-effective representation for the kernel functions. The dynamic kernel function algorithm is an extension of the fixed-point kernel function used in [50], where the unit circle of the phase function W_N is expanded by higher powers of two. The precision of the dynamic kernel function is determined by the number of bits intended to represent the twiddle factors. Two mappings, 6-bit and 10-bit, are presented here. The performances of the corresponding FFT using these dynamic kernels are compared with various hardware bit-precision considerations between FFT stages for implementation. This novel treatment attempts to bridge the gap between concept and actual implementation of the fast Fourier transform. Variable truncation scheme is also introduced to efficiently utilize the limited fixed-precision between FFT stages. The dynamic kernel function and variable truncation algorithm is developed based on the objective of accurate numerical representation and cost-effective hardware design.

4.1 Dynamic Kernel Function Algorithm

4.1.1 Algorithm

For a N-point DFT, N twiddle factors are required. These twiddle factors are N equal-distance points on the unit circle of the complex plane. In the ideal case, the real

and imaginary values for each of these kernel functions are less than one, creating an implementation challenge for hardware designers.

In the dynamic kernel function FFT [53], the kernel function is characterized by choosing a fractional representation with both the numerator and denominator chosen as an integer value to approximate the location of the twiddle factor. Once the twiddle factors for a N-point DFT is defined, the unit circle is expanded by powers of two. Then the real and imaginary portions of the twiddle factor are mapped to the closest integer values on the respective axis, representing the numerator portion of the kernel function. The number used for the scaling, the denominator portion of the kernel function, depends on the precision and hardware requirements of the FFT.

The operation of the 2-point decimation-in-frequency DFT in Fig. 4.1 is changed slightly.

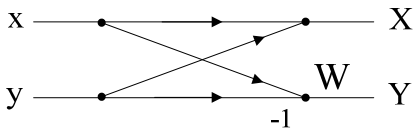

Fig. 4.1 Radix-2 DIF butterfly

Suppose the inputs x and y are 8-bits, the difference between x and y is 9-bit. The difference is "multiplied" by the numerator portion of the kernel function. Multiplication is replaced by shifts and adds operation to conserve hardware, which works fine with integer numbers. After the "multiplication" is performed, the product needs to be scaled back to 9-bits by the denominator portion of the kernel function to preserve the weighting relationships of the kernel in the FFT. The process of scaling back implies a division operation, but this is simplified to right shifts because the scaled number is power of two.

Using the dynamic kernel function algorithm for an 128-point FFT, when representing the kernel function as 6-bits, their approximated locations are shown as "*x*" versus the ideal kernel function "." in Fig. 4.2.

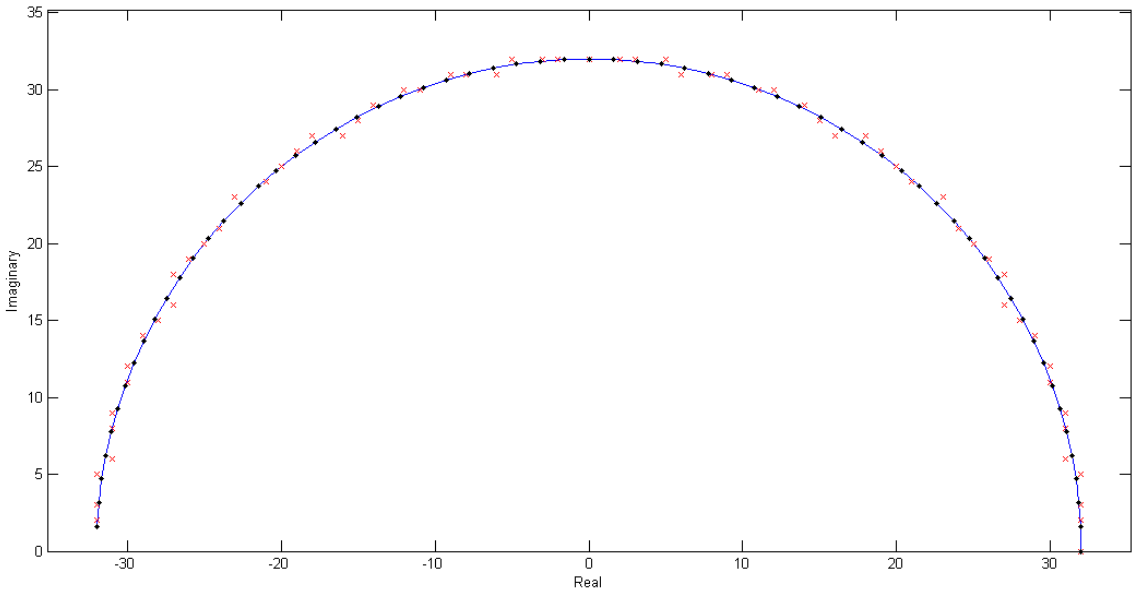

Fig. 4.2 Dynamic Kernel Function and Ideal Kernel Function Comparison (6-bit)

In comparison with the ideal kernel function with phase increments of $2\pi/N$ (or 2.8125°), the magnitude (Mag) and phase errors using dynamic kernels are shown in Table 4.1. The real and imaginary portions of the dynamic kernel function is represented as (R, I) in the table and the magnitude is scaled back for the analysis.

Table 4.1 Kernel Function Magnitude and Phase Analysis

Ideal Kernel			6-bit Dynamic Kernel			Error Difference	
Location	Phase (°)	Mag	Location	Phase (°)	Mag	Phase (°)	Mag (%)
W_N^0	0.0000	1	(32,0)	0.0000	1.0000	0.0000	0.00
W_N^1	2.8125	1	(32,2)	3.5763	1.0020	0.7638	0.19
W_N^2	5.6250	1	(32,3)	5.3558	1.0044	0.2692	0.44
W_N^3	8.4375	1	(32,5)	8.8807	1.0121	0.4432	1.21

Ideal Kernel			6-bit Dynamic Kernel			Error Difference	
Location	Phase (°)	Mag	Location	Phase (°)	Mag	Phase (°)	Mag (%)
W_N^4	11.2500	1	(31,6)	10.9541	0.9867	0.2959	1.33
W_N^5	14.0625	1	(31,8)	14.4703	1.0005	0.4078	0.05
W_N^6	16.8750	1	(31,9)	16.1892	1.0088	0.6858	0.88
W_N^7	19.6875	1	(30,11)	20.1363	0.9985	0.4488	0.15
W_N^8	22.5000	1	(30,12)	21.8014	1.0097	0.6986	0.97
W_N^9	25.3125	1	(29,14)	25.7693	1.0063	0.4568	0.63
W_N^{10}	28.1250	1	(28,15)	28.1786	0.9927	0.0536	0.73
W_N^{11}	30.9375	1	(27,16)	30.6507	0.9808	0.2868	1.92
W_N^{12}	33.7500	1	(27,18)	33.6901	1.0141	0.0599	1.41
W_N^{13}	36.5625	1	(26,19)	36.1582	1.0063	0.4043	0.63
W_N^{14}	39.3750	1	(25,20)	38.6598	1.0005	0.7152	0.05
W_N^{15}	42.1875	1	(24,21)	41.1859	0.9966	1.0016	0.34
W_N^{16}	45.0000	1	(23,23)	45.0000	1.0165	0.0000	1.65
W_N^{17}	47.8125	1	(21,41)	48.8141	0.9966	1.0016	0.34
W_N^{18}	50.6250	1	(20,25)	51.3402	1.0005	0.7152	0.05
W_N^{19}	53.4375	1	(19,26)	53.8418	1.0063	0.4043	0.63
W_N^{20}	56.2500	1	(18,27)	56.3099	1.0141	0.0599	1.41
W_N^{21}	59.0625	1	(16,27)	59.3493	0.9808	0.2868	1.92
W_N^{22}	61.8750	1	(15,28)	61.8214	0.9927	0.0536	0.73
W_N^{23}	64.6875	1	(14,29)	64.2307	1.0063	0.4568	0.63
W_N^{24}	67.5000	1	(12,30)	68.1986	1.0097	0.6986	0.97
W_N^{25}	70.3125	1	(11,30)	69.8637	0.9985	0.4488	0.15
W_N^{26}	73.1250	1	(9,31)	73.8108	1.0088	0.6858	0.88
W_N^{27}	75.9375	1	(8,31)	75.5297	1.0005	0.4078	0.05
W_N^{28}	78.7500	1	(6,31)	79.0459	0.9867	0.2959	1.33
W_N^{29}	81.5625	1	(5,32)	81.1193	1.0121	0.4432	1.21
W_N^{30}	84.3750	1	(3,32)	84.6442	1.0044	0.2692	0.44
W_N^{31}	87.1875	1	(2,32)	86.4237	1.0020	0.7638	0.19
W_N^{32}	90.0000	1	(0,32)	90.0000	1.0000	0.0000	0.00
W_N^{33}	92.8125	1	(-2,32)	93.5763	1.0020	0.7638	0.19
W_N^{34}	95.6250	1	(-3,32)	95.3558	1.0044	0.2692	0.44
W_N^{35}	98.4375	1	(-5,32)	98.8807	1.0121	0.4432	1.21
W_N^{36}	101.2500	1	(-6,31)	100.9541	0.9867	0.2959	1.33
W_N^{37}	104.0625	1	(-8,31)	104.4703	1.0005	0.4078	0.05
W_N^{38}	106.8750	1	(-9,31)	106.1892	1.0088	0.6858	0.88
W_N^{39}	109.6875	1	(-11,30)	110.1363	0.9985	0.4488	0.15
W_N^{40}	112.5000	1	(-12,30)	111.8014	1.0097	0.6986	0.97
W_N^{41}	115.3125	1	(-14,29)	115.7693	1.0063	0.4568	0.63
W_N^{42}	118.1250	1	(-15,28)	118.1786	0.9927	0.0536	0.73
W_N^{43}	120.9375	1	(-16,27)	120.6507	0.9808	0.2868	1.92
W_N^{44}	123.7500	1	(-18,27)	123.6901	1.0141	0.0599	1.41
W_N^{45}	126.5625	1	(-19,26)	126.1582	1.0063	0.4043	0.63
W_N^{46}	129.3750	1	(-20,25)	128.6598	1.0005	0.7152	0.05
W_N^{47}	132.1875	1	(-21,24)	131.1859	0.9966	1.0016	0.34

Ideal Kernel			6-bit Dynamic Kernel			Error Difference	
Location	Phase (°)	Mag	Location	Phase (°)	Mag	Phase (°)	Mag (%)
W_N^{48}	135.0000	1	(-23,23)	135.0000	1.0165	0.0000	1.65
W_N^{49}	137.8125	1	(-24,21)	138.8141	0.9966	1.0016	0.34
W_N^{50}	140.6250	1	(-25,20)	141.3402	1.0005	0.7152	0.05
W_N^{51}	143.4375	1	(-26,19)	143.8418	1.0063	0.4043	0.63
W_N^{52}	146.2500	1	(-27,18)	146.3099	1.0141	0.0599	1.41
W_N^{53}	149.0625	1	(-27,18)	149.3493	0.9808	0.2868	1.92
W_N^{54}	151.8750	1	(-28,15)	151.8214	0.9927	0.0536	0.73
W_N^{55}	154.6875	1	(-29,14)	154.2307	1.0063	0.4568	0.63
W_N^{56}	157.5000	1	(-30,12)	158.1986	1.0097	0.6986	0.97
W_N^{57}	160.3125	1	(-30,11)	159.8637	0.9985	0.4488	0.15
W_N^{58}	163.1250	1	(-31,9)	163.8108	1.0088	0.6858	0.88
W_N^{59}	165.9375	1	(-31,8)	165.5297	1.0005	0.4078	0.05
W_N^{60}	168.7500	1	(-31,6)	169.0459	0.9867	0.2959	1.33
W_N^{61}	171.5625	1	(-32,5)	171.1193	1.0121	0.4432	1.21
W_N^{62}	174.3750	1	(-32,3)	174.6442	1.0044	0.2692	0.44
W_N^{63}	177.1875	1	(-32,2)	176.4237	1.0020	0.7638	0.19

When compared with the ideal kernel, the dynamic kernel function has a worst case magnitude difference of about 2% with a 1° disparity in the phase.

For the digital wideband receivers in [48-50], the FFT implementation used fixed-point kernel function FFT as shown in Fig. 4.3. The Monobit receivers "fixed" the kernel function to just twelve points for the entire FFT operation which amplified the approximation error with an increase in DFT size.

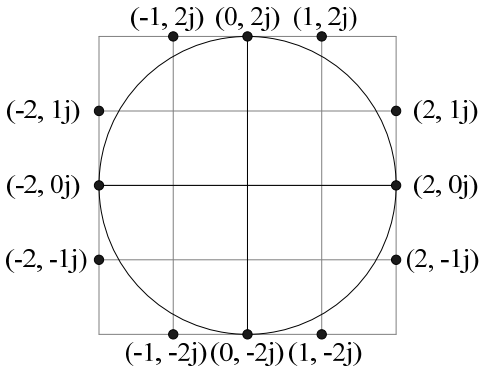

Fig. 4.3 Fixed-point kernel function

Since the radius of the unit circle is one, the error may be deduced by taking the square root of the sum of squares for the real and imaginary components. In comparison with the ideal kernel function W_N^{16} for representing the angle at π/4, regardless of using either (2, j) or (1, 2j), 45° the fixed-kernel implementation, (2, j) = 26.5651°, yields an error of 18.4° in phase. This is much worse than the worst case scenario of the dynamic kernel function. From Figs. 4.2 and 4.3, it is shown that the rounding error is much less in the dynamic kernel function than the fixed-point kernel function.

4.1.2 Bit Precision

An obvious observation is that the unit circle scaled with a higher power of two will have a better approximation of the kernel function. Utilizing the dynamic kernel function algorithm, a 10-bit kernel is used to assess the relevant performance metrics with the 6-bit kernel shown in the previous section. For a 10-bit kernel, their approximated locations are shown as "*x*" versus the ideal kernel function "." in Fig. 4.4.

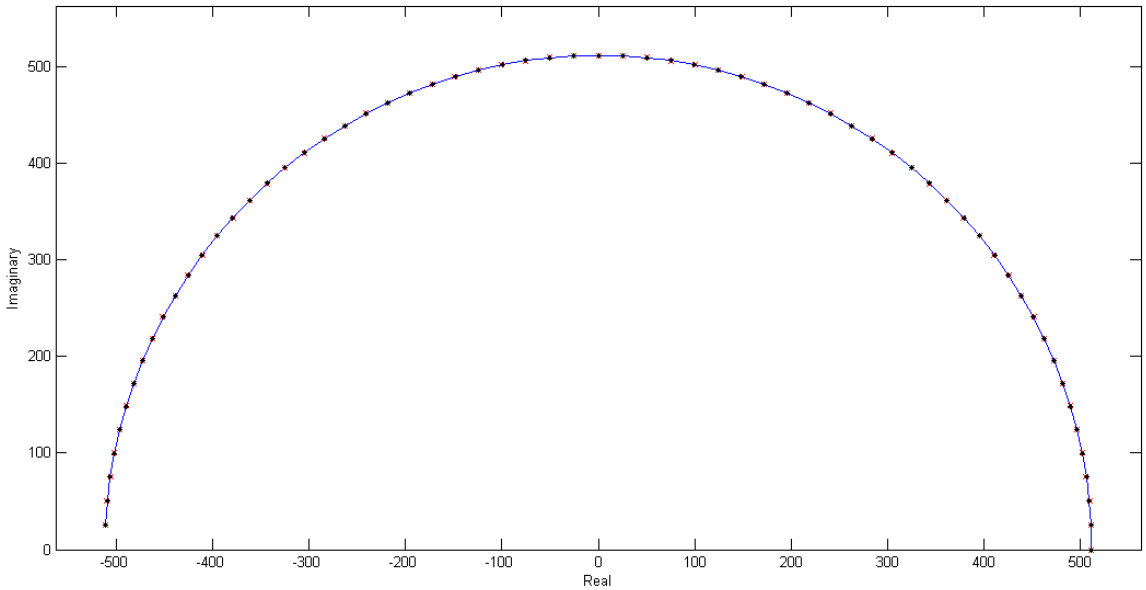

Fig. 4.4 Dynamic Kernel Function and Ideal Kernel Function Comparison (10-bit)

When compared with the ideal kernel, the 10-bit dynamic kernel function has a worst case magnitude difference of about 0.1% with a 0.06° disparity in the phase. A further improvement when assessed with the 6-bit representation. The only draw back is the significant increase when changing the kernel function from 6 to 10 bits.

4.2 FFT Word Length Consideration

4.2.1 FFT Word Length

In the ideal situation, the bit precision for each data would gradually increase as the sampled data progress through the arithmetic functions of the FFT. This fact is illustrated for an 128-point FFT with 8-bit inputs in Fig. 4.5. A minimum of one additional bit is needed for each stage of the FFT to prevent overflow.

Fig. 4.5 Ideal Word Lengths for 128-Point FFT Implementation

However, when implemented in hardware, the target implementation platform may not have enough resources to accommodate the large memory requirements needed for the last few stages of the FFT. Most designers would then constrain the inter-stage precision to a fixed number of bits. The conventional approach is to take the most significant bits (MSB) after the butterfly arithmetic calculations are performed. For a design with an 8-bit fixed-precision, the 9-bits outputs of each stage will constantly need to truncate away 1 LSB bit, as shown in Fig. 4.6.

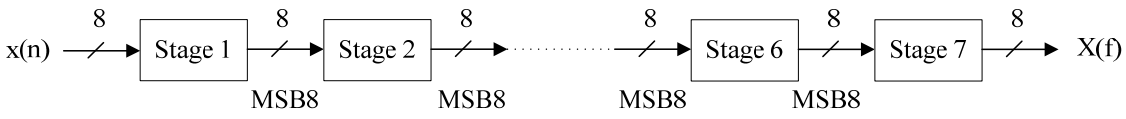

Fig. 4.6 Conventional design for 128-Point FFT Implementation

This will suit well when considering only the high and moderately high input amplitude signals, but it leads to detection loss of weak signals. For weak signals, such as a sampled signal which may only occupy 4 out of 8-bits of the ADC, the important information bits will be truncated away with the first few stages which may also introduce error. The errors will be propagated and accumulated as the data progressed through the FFT, leading to high noise spurs and signal detection loss in the frequency spectrum.

4.2.2 Variable Truncation Scheme (VTS)

A variable truncation scheme is presented to amend the above predicament as well as keeping a low hardware usage. While maintaining the current hardware design for the DFT butterflies accepting an 8-bit value and outputting a 9-bit value, extra comparators are placed at the outputs of the butterflies. These comparators will determine whether the 9-bit two's complement values exceeds the absolute value of 128, the full scale value of an 8-bit number, and generates a status flag. The status flags controls the subsequent stage to take the MSB 8-bits when the value from preceding stage exceeds 128, and the eight least significant bits (LSB) when it is less or equal. The resulting datapath is presented in Fig. 4.7. The shaded areas represent the comparison logic to perform variable truncation.

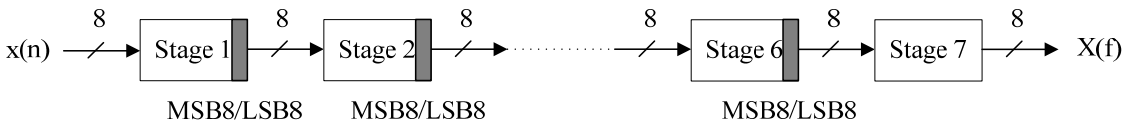

Fig. 4.7 128-Point FFT Implementation with Variable Truncation

This scheme limits the noise spurs generated due to truncation while preserving the magnitude of the frequency bin representing the weak signal. The scheme may be applied for every stage of the FFT, effectively minimizing the error introduced from truncation for fixed-precision while lowering the hardware cost increase in the later stages of the transform.

4.2.3 Multiple N-bit Input Selections (MIS)

Variable truncation may also be applied to the input of the FFT. In the case of our research group, a high-speed ADC on the prototyping board constantly feeds 10-bit data to the Xilinx FPGA. Unfortunately, we were unable to fit a FFT design with a 10-bit input and inter-stage precision to process the full scale range of the ADC. The decision was made to design a 128-point FFT with 8-bit inputs and inter-stage precision to accept the most significant 8-bits from the ADC. The name multiple N-bit input selection (MIS) is used here to differential between the variable truncation used before the FFT from the one applied between the FFT stages. The algorithm used is similar to variable truncation scheme.

Maintaining the same 128-point FFT with 8-bit inputs and bit precision, variable truncation scheme may be applied after the initial data collection of 128 points for FFT processing. The full 10-bit 2's complement number, A[9:0], from the ADC are collected, and variable truncation scheme may be applied to choose the various 8-bit input

combinations, as described in Table 4.2. The table is not exhaustive and may be extended for weaker signals. A status flag is needed and a priority must be set to accommodate the largest number within each 128 data collected. For any value that has a 10 or 01 combinations in the MSB 2 bits, the MSB 8 bits A[9:2] must be used.

Table 4.2 Multiple N-bit Input Selections From 10-bits (N=8)

10-bit Input A[9:0]										MIS Selection
A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	
1	0	x	x	x	x	x	x	x	x	A[9:2]
0	1	x	x	x	x	x	x	x	x	
1	1	x	x	x	x	x	x	x	x	A[8:1]
0	0	x	x	x	x	x	x	x	x	
1	1	1	x	x	x	x	x	x	x	A[7:0]
0	0	0	x	x	x	x	x	x	x	
1	1	1	1	x	x	x	x	x	x	A[6:0] & 1’b0
0	0	0	0	x	x	x	x	x	x	
1	1	1	1	1	x	x	x	x	x	A[5:0] & 2’b00
0	0	0	0	0	x	x	x	x	x	
1	1	1	1	1	1	x	x	x	x	A[4:0] & 3’b000
0	0	0	0	0	0	x	x	x	x	

For 10-bit inputs with four consecutive ones or zeros in the MSB 4 bits, the LSB 7 bits are taken from the sampled input, and one bit with a value zero is padded at the end. This ensures the full utilization of the 8-bit FFT input by amplifying the sampled data to prevent constant truncation of critical information bits after every FFT stage when using the conventional approach for dealing with inter-stage bit precision as shown in Fig. 4.6.

4.3 Performance Analysis

Using an 128-point FFT, a series of case studies are performed using 6-bit and 10-bit dynamic kernels using the different variations of inter-stage bit precision implementation shown in Figs. 4.5, 4.6, 4.7. Since the target FFT design is used for

multi-tone signal detection, the effects on the addition of a Hamming window function are also simulated.

4.3.1 Test Setup and Input Stimulus

The common parameters for the analysis performed are listed below.

1. 128-point DIF FFT with 8-bit inputs is used.
2. The first and last three bins in the frequency spectrum are omitted. (Aliasing)
3. The sampling frequency is 2.048 GHz, thus the frequency resolution $\Delta f = 16$ MHz.
4. Input signal-to-noise ratio is varied from 0 to 100 dB in 5 dB increments for each frequency test. Finer increment of 1 dB is used for SNR from 0 to 30 dB.
5. The input frequency is swept from 48 MHz to 960 MHz in 16 MHz intervals. (On bin)
6. 100 runs are performed for each input frequency with random Gaussian noise and phase.
7. The averaged magnitude, phase error, and SFDR of the primary signal in the frequency spectrum are recorded.
8. The $R^2 + I^2$ values of each frequency bin is used for implementation analysis.
9. The smallest $R^2 + I^2$ value allowed is 1. This prevents small decimal numbers between 0 and 1 causing havoc on the SFDR value when converted to dB.
10. (58 Frequency Values) x (100 runs per frequency) x (45 SNR values) = 261,000 single-tone signal test cases are performed for each case study.

For the simulated analog inputs, the signals are scaled to 8-bits to maximize the expected SFDR performance. The following pseudo code shows the procedures for RF signal generation *(RF_input)* for each test.

1. Input frequency *fin* and SNR are known.
2. Phase *phi* is generated based on a uniform distribution between $-\pi$ to π.
3. Noise is generated using the randn function. Noise = randn(1, fft_point)
4. input = cos (2π fin + *phi*)
5. Current input and noise has amplitudes of 1.
6. The noise amplitude *noise_amp* is computed using the defined SNR value.
   noise_amp = $10^{-(SNR/20)}$
7. test_input = input + noise_amp*noise.
8. test_input is normalized with respect to the largest amplitude.
9. test_input is scaled to 127, the full scale value of an 8-bit 2's complement number.
10. RF_input = floor(test_input) to simulate the ADC behavior.

To simulate the impact of replacing the ideal kernel with dynamic kernel function, the kernel function in each stage is replaced by the corresponding hardware implementation of the dynamic kernel progressively. Thus eight different scenarios, m0 to m7 are tested. m0 denotes all stages are using ideal kernels; m1 denotes the 1st stage of the FFT uses the dynamic kernel function while the remaining stages still uses the ideal kernel function; m2 denotes the first two stages of the FFT are replaced by the dynamic kernel function; finally, m7 denotes a full dynamic kernel function FFT.

4.3.2 Continual Word Length Growth

The ideal implementation scenario in Fig. 4.5 is first tested with continual word length growth as the control case of the experiment. The SFDR with 6-bit and 10-bit dynamic kernel functions are shown in Figs. 4.8 and 4.9.

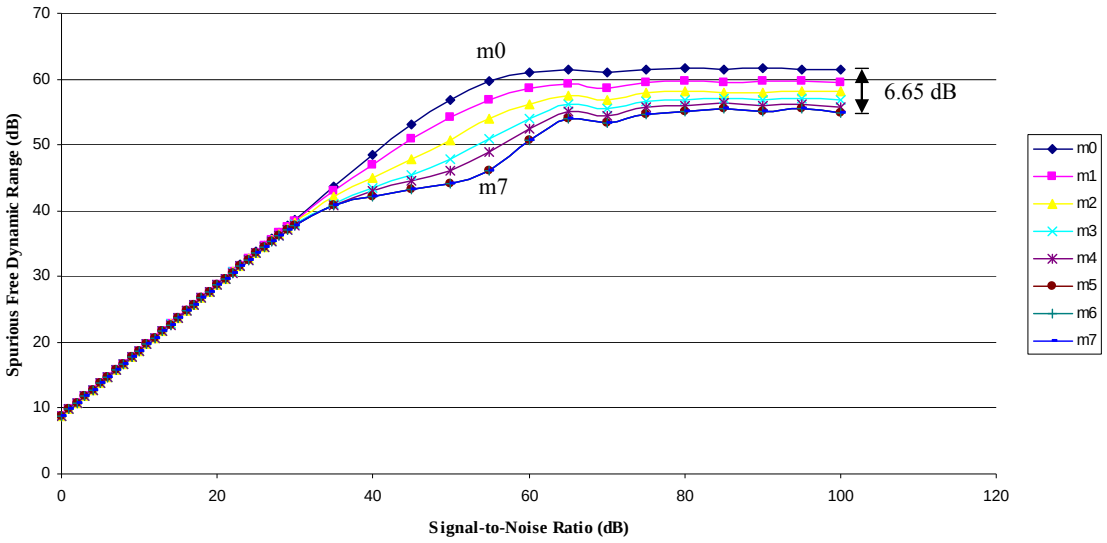

Fig. 4.8 SFDR vs. SNR Performance (6-bit Dynamic Kernel, No Windowing)

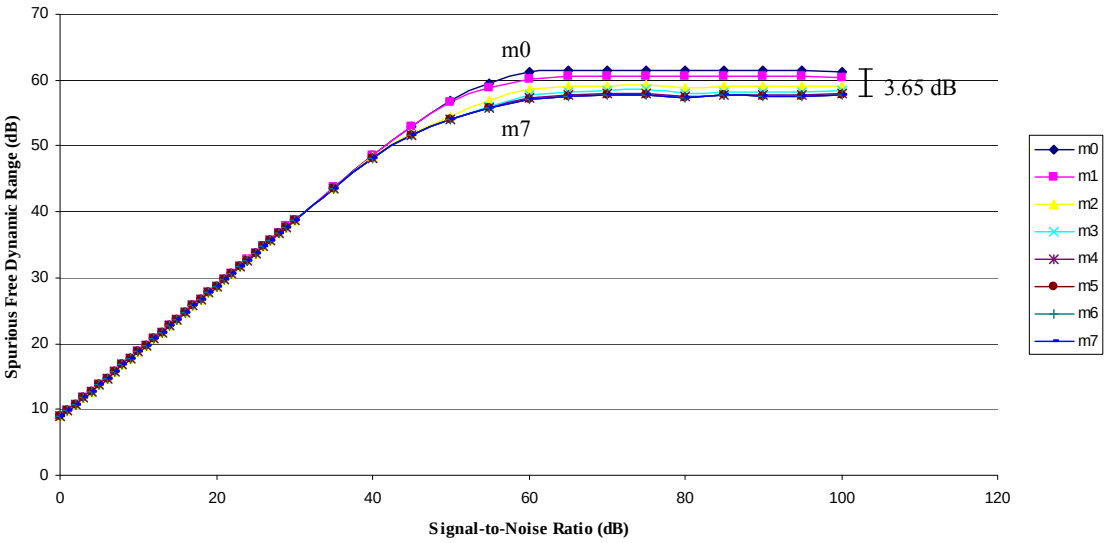

Fig. 4.9 SFDR vs. SNR Performance (10-bit Dynamic Kernel, No Windowing)

For the 6-bit dynamic kernel, there is a 6.65 dB difference between using full ideal and hardware kernels, m0 and m7, in a low noise environment (SNR = 100 dB). Note that the SFDR is almost the same for the ideal and 6-bit kernels from SNR of 0 to 30 dB, then the noise spurs eventually drops lower than the second harmonics of the primary signal. The maximum expected SFDR performance for hardware implementation is around 54.6 dB.

When using the 10-bit kernel, there is only a SFDR difference of 3.65 dB between ideal and dynamic kernels at SNR of 100 dB, a 3 dB improvement over the 6-bit kernel. The expected SFDR performance saturates at 57.6 dB, also a 3 dB improvement. The SFDR of the 10-bit kernel remains the same as the ideal from 0 to 40 dB, a wider range (10 dB more) in comparison with the one of 6-bit kernel for SNR from 0 to 30 dB.

The estimated phase error (in degrees) of the input signal based on the real and imaginary values of detected primary signal in the frequency spectrum for both 6-bit and 10-bit kernels are almost identical. For SNR of 0 to 30 dB, the phase performance of the 6-bit kernel is presented in Fig. 4.10.

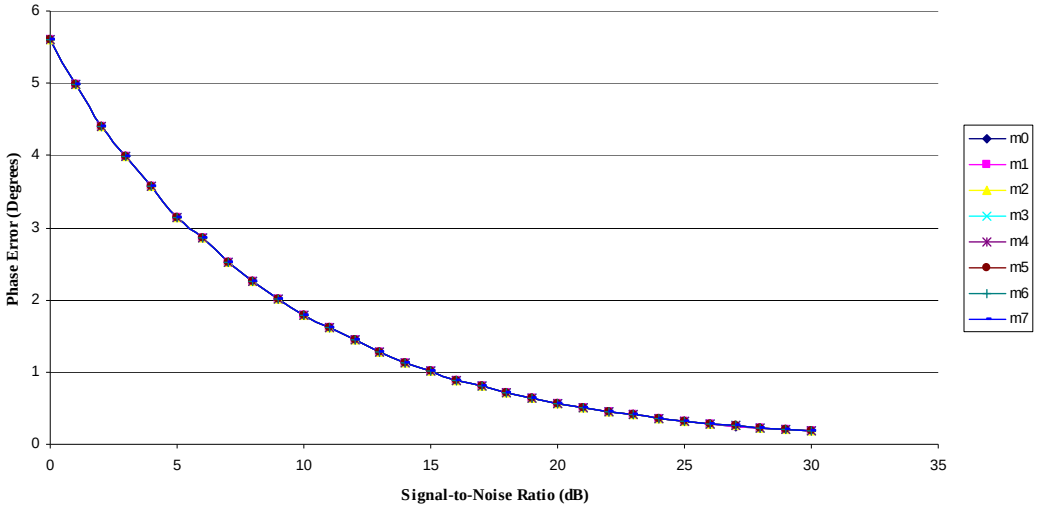

Fig. 4.10 Phase Error vs. SNR (6-bit Dynamic Kernel, No Windowing)

For SNR of 0 to 30 dB, there is no observable phase estimation difference between the ideal and 6-bit kernel. The phase estimation error for 6-bit and 10-bit kernels is also similar for SNR from 30 to 100 dB, thus only the result from the 6-bit kernel is plotted in Fig. 4.11. There is only a 0.004 degrees difference between m0 and m7.

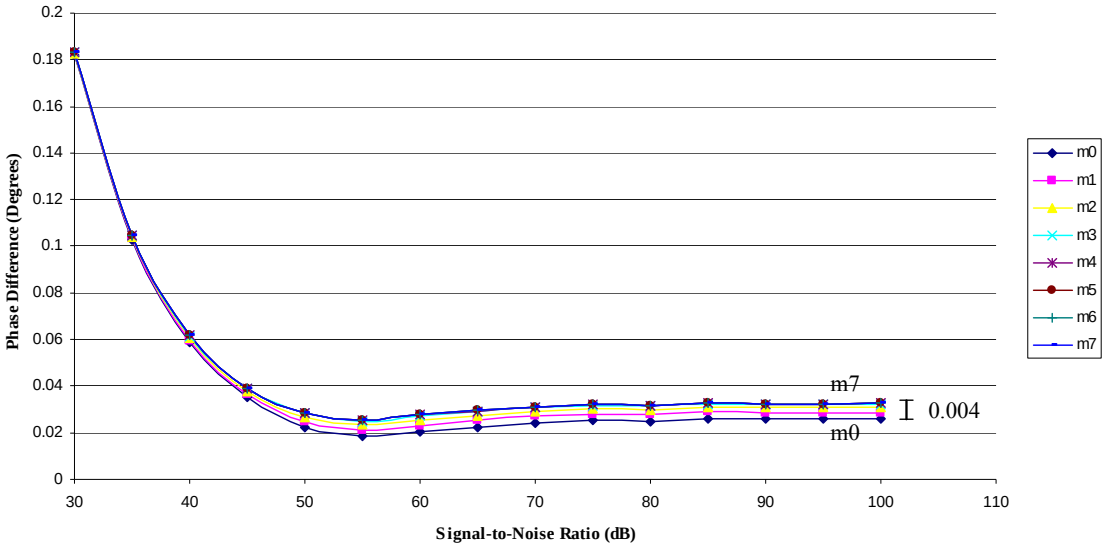

Fig. 4.11 Phase Error vs. SNR (6-bit Dynamic Kernel, No Windowing)

The SFDR performances with the application of Hamming window function are presented in Figs. 4.12 and 4.13 for 6-bit and 10-bit kernels. The SFDR of the ideal kernel lost about 7.5 dB when compared with no window function applied. The best case SFDR of the 6-bit kernel dropped to 41.9 dB from 54.6 dB, and 57.6 dB to 46.7 dB for the 10-bit kernel. The 10-bit kernel experienced less SFDR loss.

The phase estimation error for 6-bit and 10-bit kernel after the window function for SNR of 0 to 30 dB is very similar to Fig. 4.10, but the error is 1 degree worse than before.

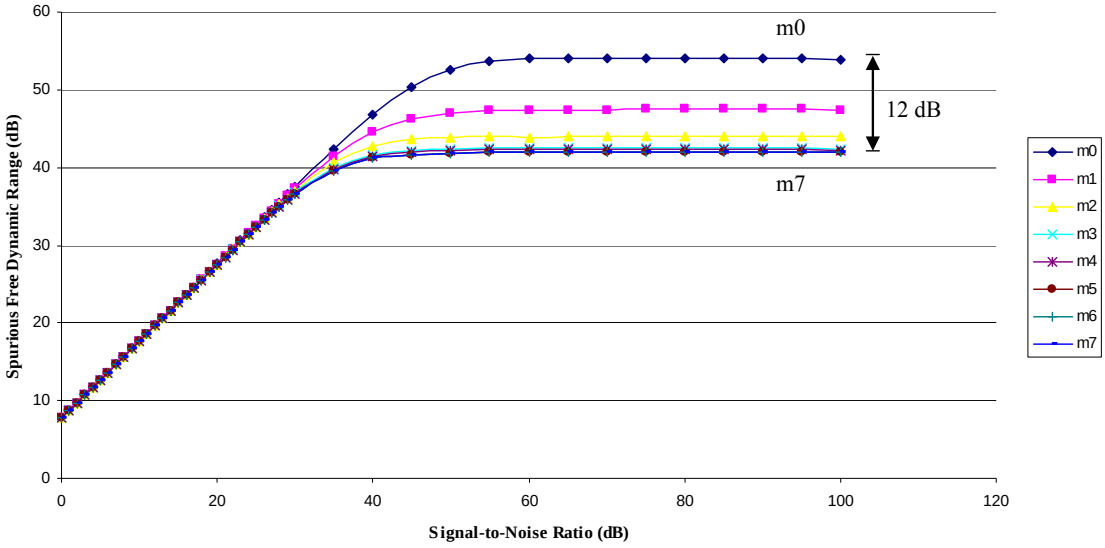

Fig. 4.12 SFDR vs. SNR Performance (6-bit Dynamic Kernel, Hamming Window)

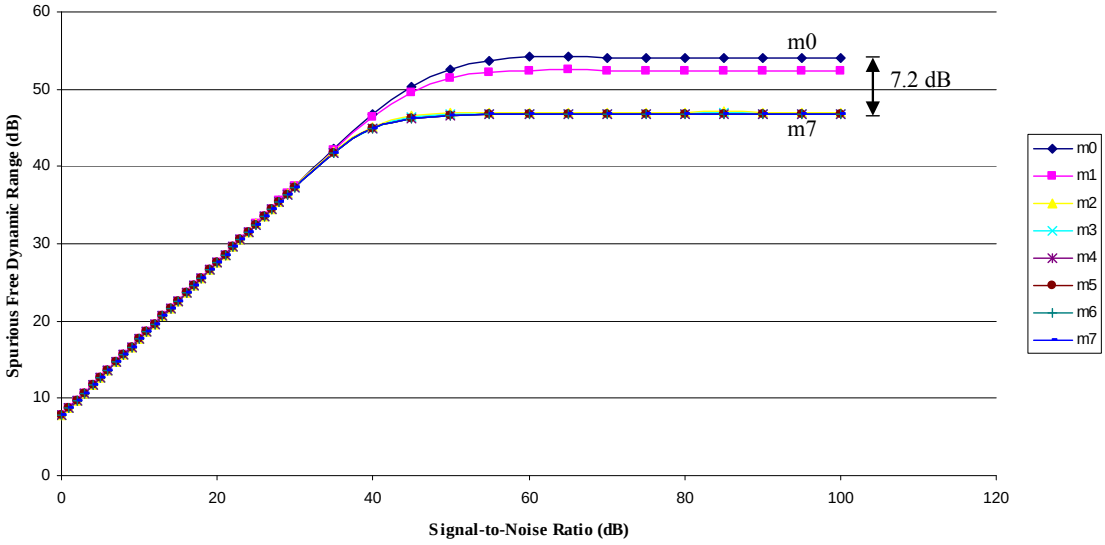

Fig. 4.13 SFDR vs. SNR Performance (10-bit Dynamic Kernel, Hamming Window)

For SNR ranging from 30 to 100 dB, the phase estimation is similar for 6-bit and 10-bit kernel. However, the phase estimation error between the ideal and the 6-bit hardware kernels has increased to 0.014 degree difference, shown in Fig. 4.14.

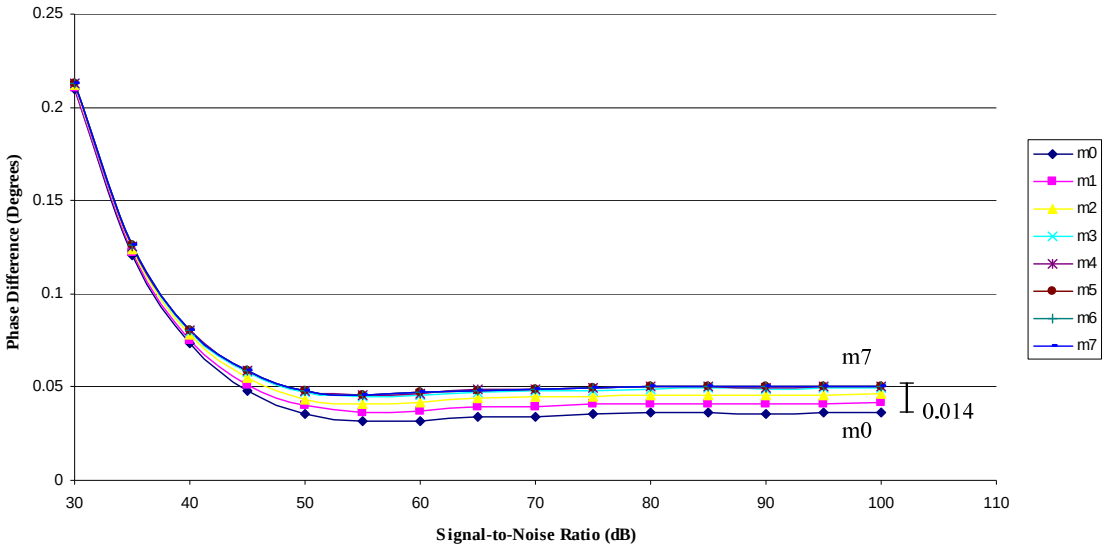

Fig. 4.14 Phase Error vs. SNR (6-bit Dynamic Kernel, Hamming Window)

4.3.3 Constant Word Length Truncation

This section models the performance of the conventional approach to designing FFT with fixed-precision between FFT stages as shown in Fig. 4.6. The SFDR with 6-bit and 10-bit dynamic kernel functions are shown in Figs. 4.15 and 4.16.

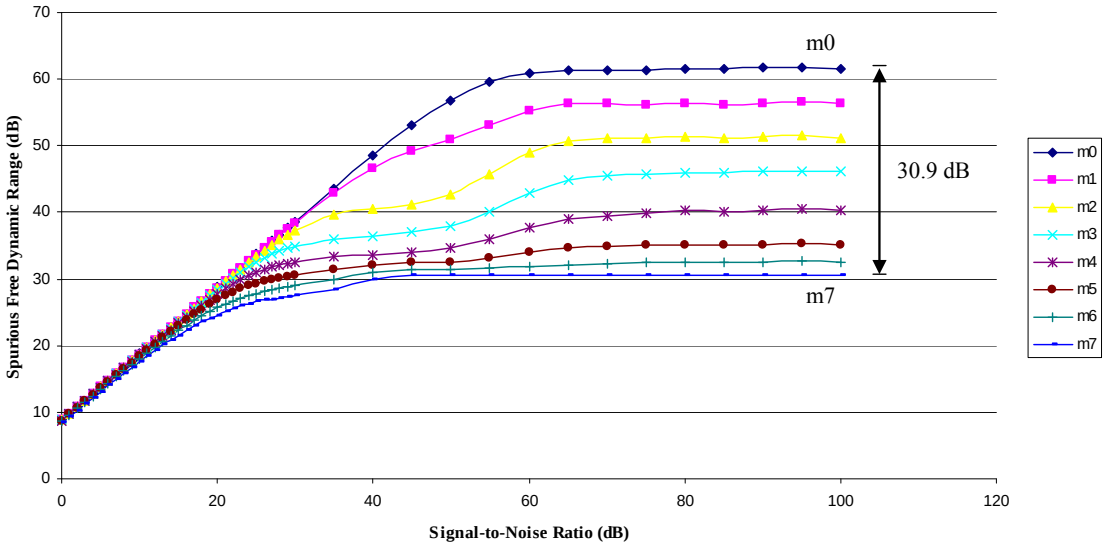

Fig. 4.15 SFDR vs. SNR Performance (6-bit Dynamic Kernel, No Window)

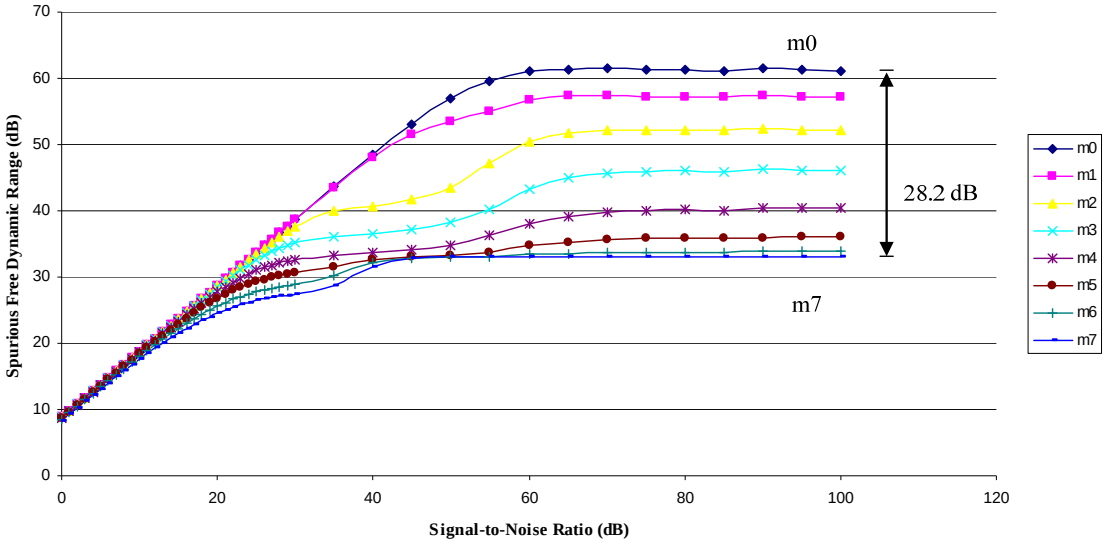

Fig. 4.16 SFDR vs. SNR Performance (10-bit Dynamic Kernel, No Window)

In both 6-bit and 10 bit kernels with constantly truncating away 1 LSB bit away, approximately 30 dB are lost when compared with the ideal kernel. The best SFDR performance for the 6-bit kernel dropped from 54.6 dB to 30.5 dB, and similar losses are found for the 10-bit kernel (57.6 dB to 33 dB). The phase estimation error of 6 bit kernel are illustrated in Figs. 4.17 and 4.18.

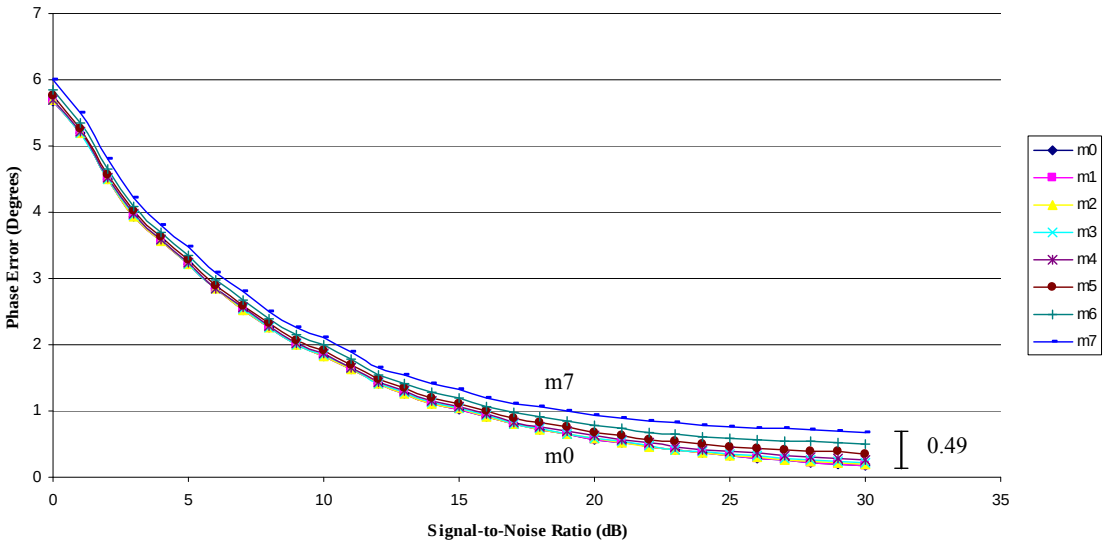

Fig. 4.17 Phase Error vs. SNR (6-bit Dynamic Kernel, No Windowing)

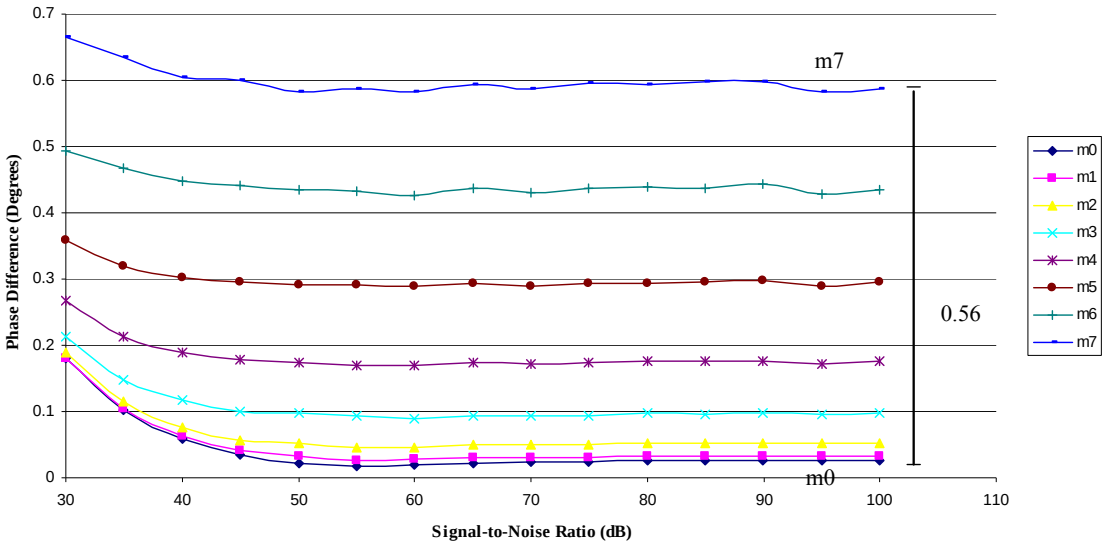

Fig. 4.18 Phase Error vs. SNR (6-bit Dynamic Kernel, No Windowing)

A phase difference of 0.49 degrees is found between the ideal and 6-bit kernel function in Fig. 4.17. In general, the phase estimation errors are worse than the FFT with continual word length growth, but phase errors are still within 1 degree for SNR of 18-100 dB. The 6-bit and 10-bit kernel has comparable phase estimation errors thus the phase estimation plots for 10-bit kernel are omitted.

The SFDR of 6-bit and 10-bit kernel with Hamming window are shown in Figs. 4.19 and 4.20. When evaluated against the FFT with the same setup but with continually growing word length (Figs. 4.12 and 4.13), the expected SFDR performance for 6-bit kernel function lost further ground from 41.9 dB to 26 dB. The 10-bit kernel SFDR dropped from 46.7 dB to 27.3 dB. Different from the last case study, the SFDR for SNR of 0 to 30 dB are no longer the same between the ideal and dynamic kernel function.

The phase estimation error with Hamming window seems to be the same between the 6-bit and 10-bit kernels. From this point on, only the phase estimation error of the 6-bit kernel will be plotted.

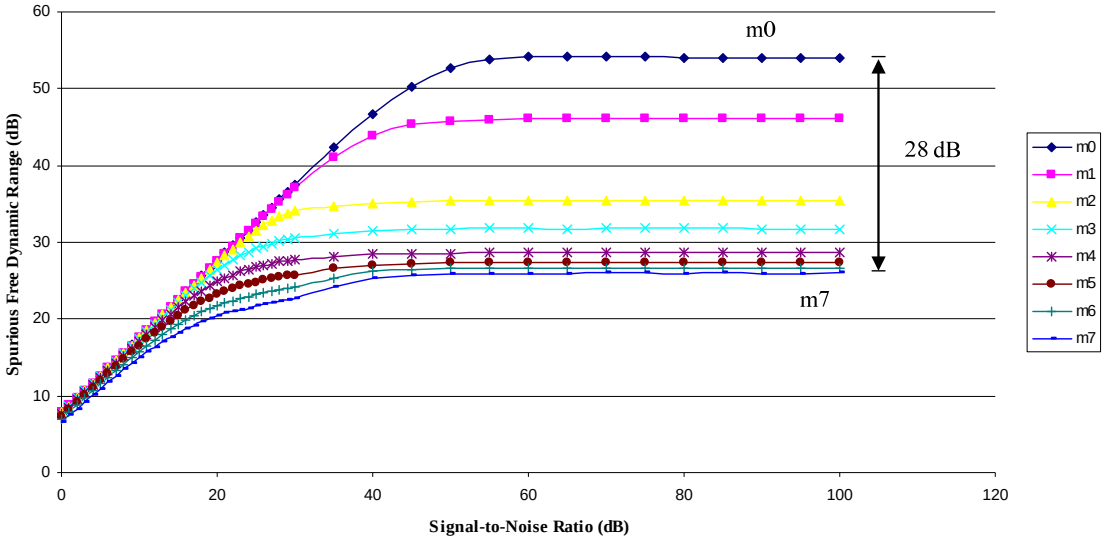

Fig. 4.19 SFDR vs. SNR Performance (6-bit Dynamic Kernel, Hamming Window)

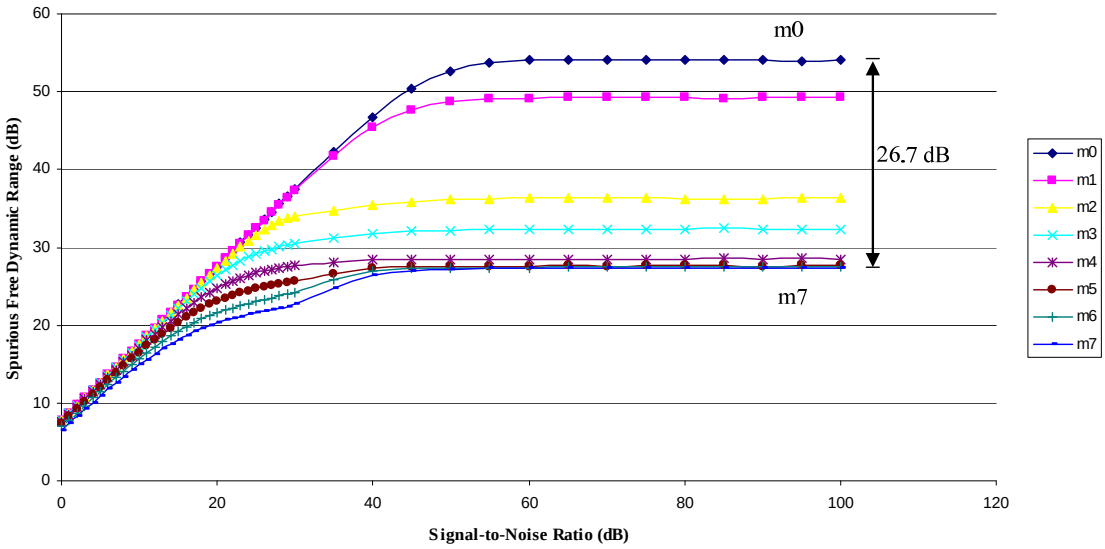

Fig. 4.20 SFDR vs. SNR Performance (10-bit Dynamic Kernel, Hamming Window)

The phase estimation error using the conventional design approach with the Hamming window function is shown in Figs. 4.21 and 4.22. There seems to be a consistent one degree phase error when compared against the ideal case. The phase errors are less than 2 degrees for SNR values greater than 15 dB.

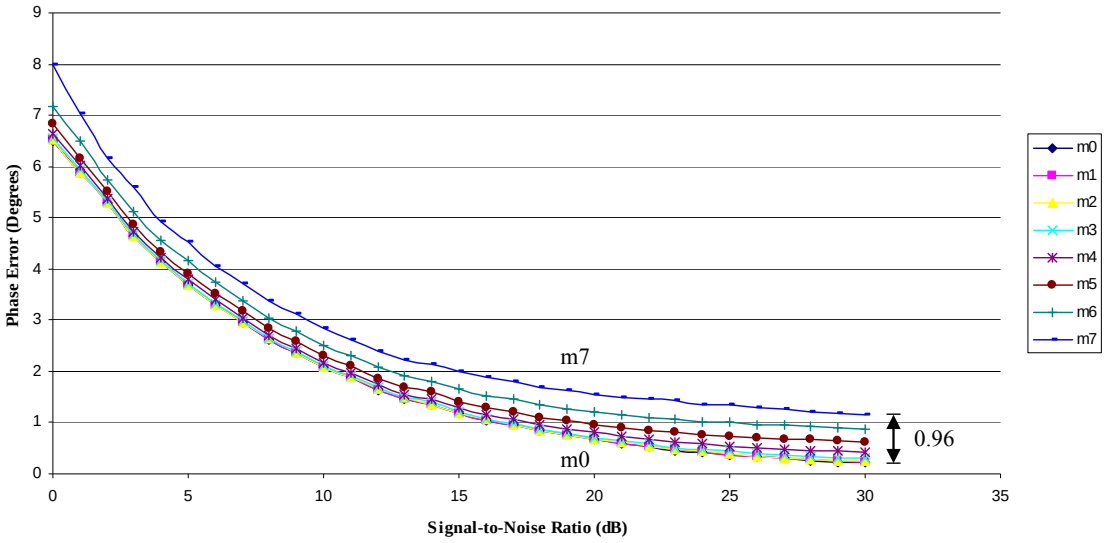

Fig. 4.21 Phase Error vs. SNR (6-bit Dynamic Kernel, Hamming Window)

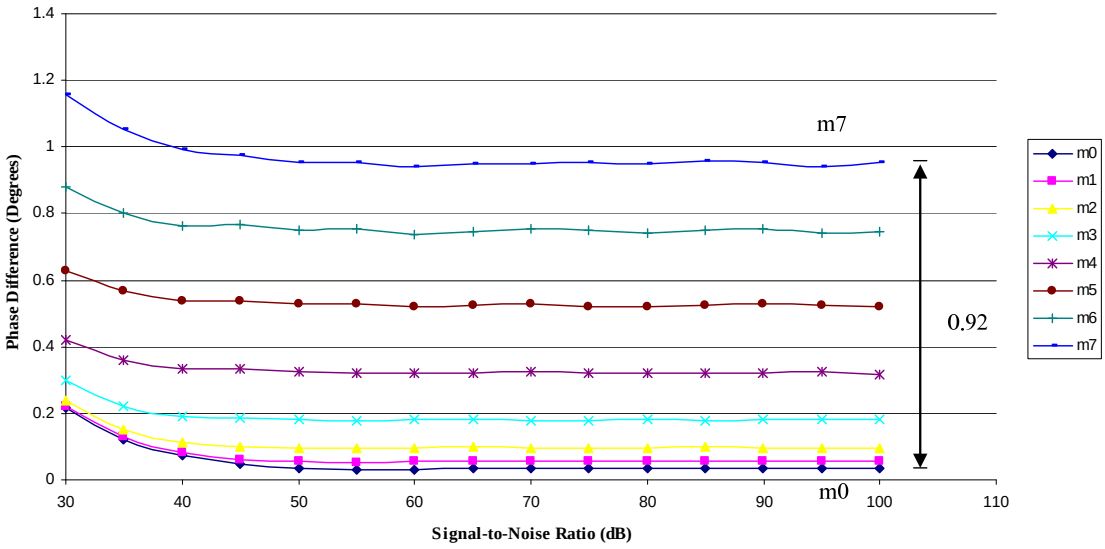

Fig. 4.22 Phase Error vs. SNR (6-bit Dynamic Kernel, Hamming Window)

4.3.4 Constant Word Length with Variable Truncation

The last set of performance assessment uses the variable truncation scheme presented in section 4.2.2 with 6-bit and 10-bit dynamic kernels. The SFDR performances are presented in Figs. 4.23 and 4.24 without windowing.

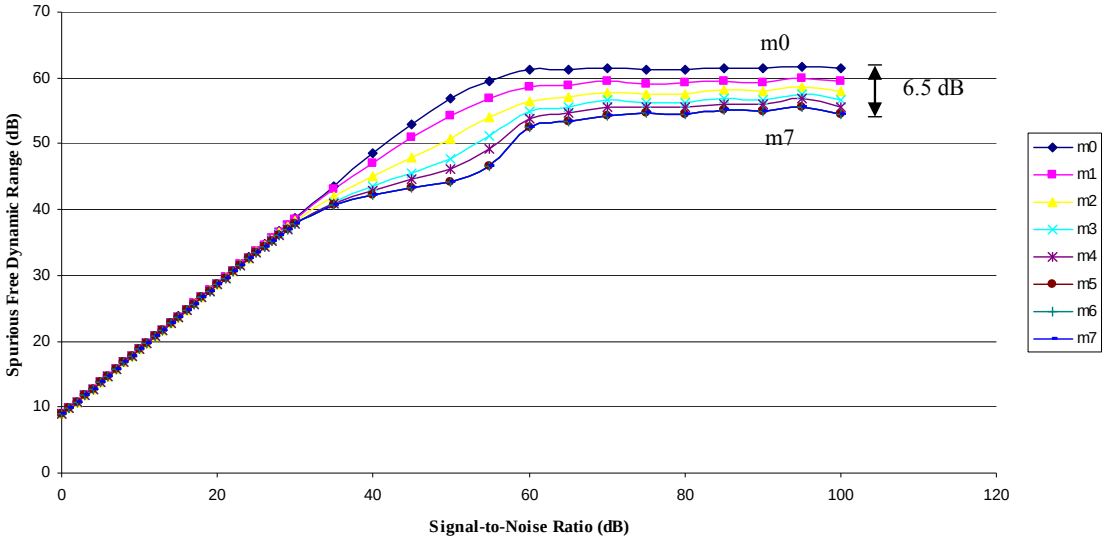

Fig. 4.23 SFDR vs. SNR Performance (6-bit Dynamic Kernel, No Window)

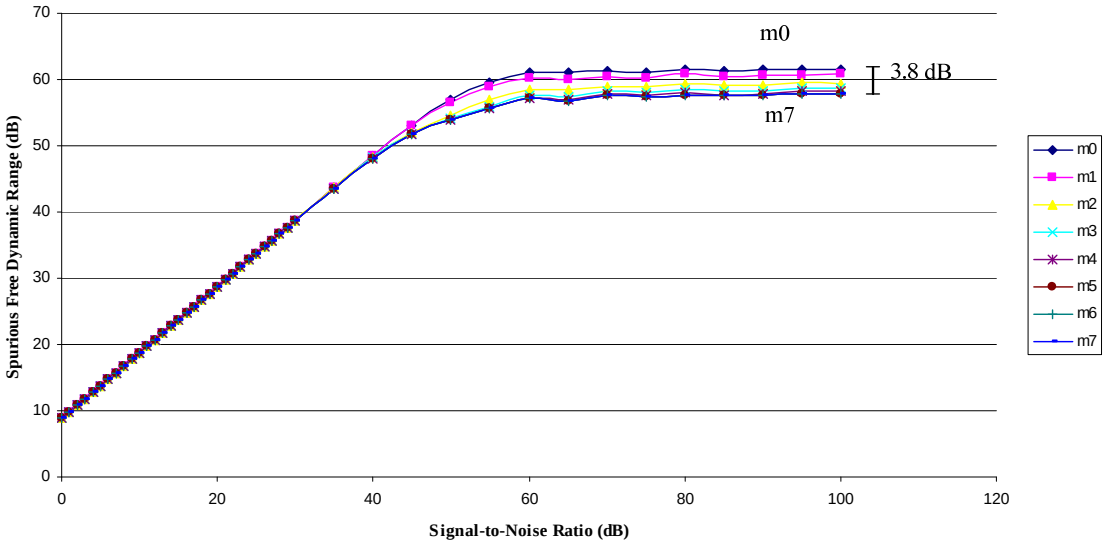

Fig. 4.24 SFDR vs. SNR Performance (10-bit Dynamic Kernel, No Window)

In comparison with the ideal implementation, the SFDR performances using variable truncation scheme has about the same performance for both 6-bit and 10-bit kernel implementation when compared side-by-side with the constantly growing word length FFT. The phase error estimation plot also looks exactly the same as those presented in section 4.3.3. The SFDR performances with Hamming window function are plotted in Figs 4.25 and 4.26.

Figs. 4.25 and 4.26 look very similar to Figs. 4.12 and 4.13. This indicates that for a single-signal, an 8-bit precision is sufficient to maintain the SFDR of the system. In Fig. 4.5, the last stage of the FFT has 15-bit outputs compared with the 8-bit outputs of variable truncation scheme in Fig. 4.7.

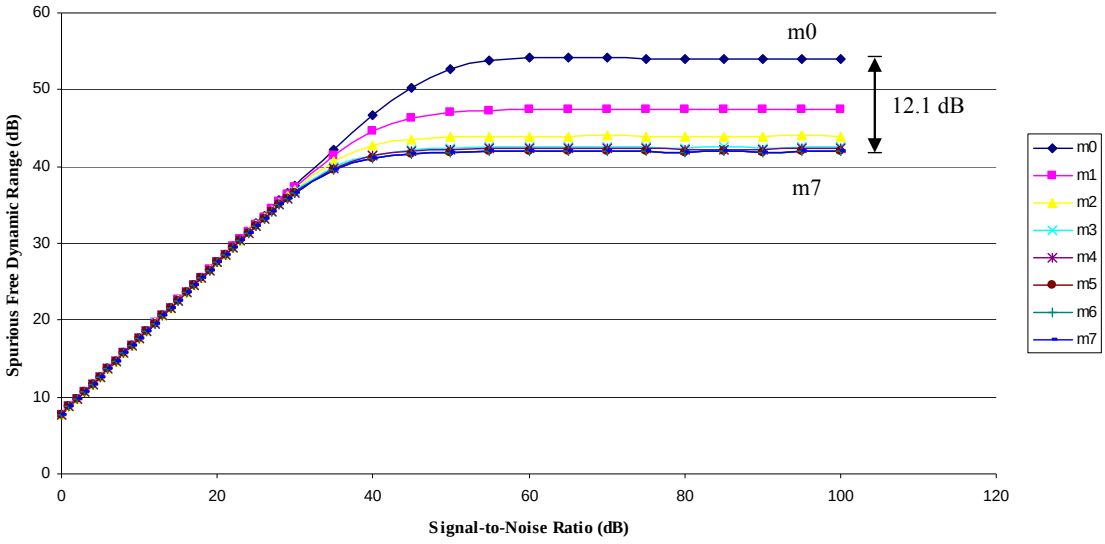

Fig. 4.25 SFDR vs. SNR Performance (6-bit Dynamic Kernel, Hamming Window)

For FFT with higher lengths, variable truncation scheme shows a great potential in minimizing the word length required for the later stages of the FFT. However, there are still drawbacks to the design. In order to increase the IDR for multi-tone signals, the

bit precision between FFT stages must be allowed to grow. This is the area where the hardware designer must make a decision based on the requirements of the FFT processor.

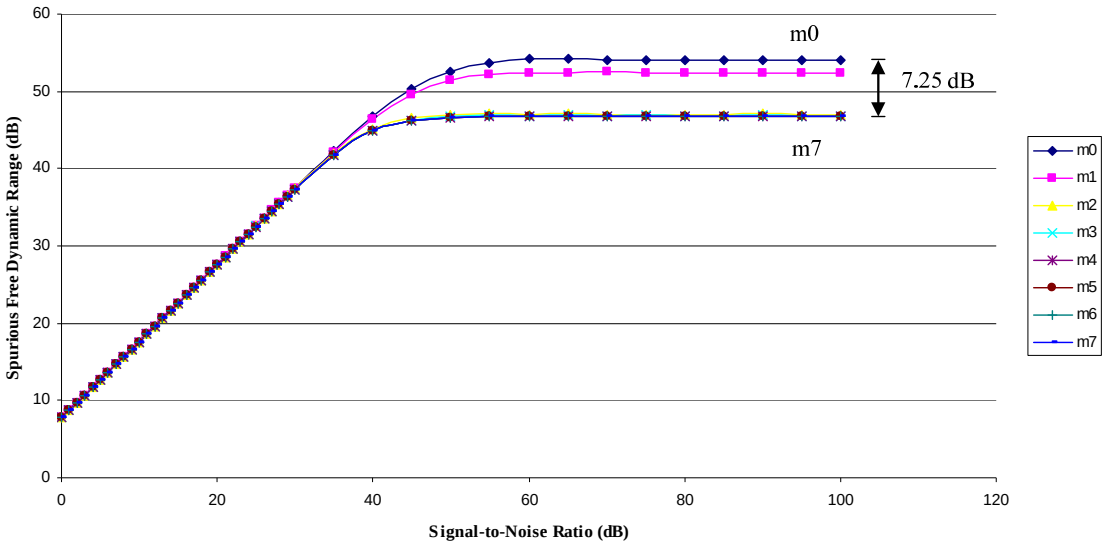

Fig. 4.26 SFDR vs. SNR Performance (10-bit Dynamic Kernel, Hamming Window)

The above analysis can be summed up in Tables 4.3 and 4.4, which shows promising results for minimal SFDR difference between ideal and dynamic kernel function as well as maximum expected SFDR using dynamic kernel function.

Table 4.3 Ideal Versus Dynamic Kernel Function SFDR Difference

Wordlength Consideration	No Window		Hamming Window	
	6-bits	10-bits	6-bits	10-bits
Continuous Growth	6.65 dB	3.65 dB	12 dB	7.2 dB
Constant Truncation	30.9 dB	28.2 dB	28 dB	26.7 dB
Variable Truncation Scheme	6.5 dB	3.8 dB	12.1 dB	7.25 dB

The estimated SFDR performance using variable truncation scheme after each stage will reach about the same level of performance as the implementation with constant

wordlength growth without increasing the number of bits for the arithmetic computations in the later stages.

Table 4.4 Maximum Expected SFDR Using Dynamic Kernel Function

Wordlength Consideration	No Window		Hamming Window	
	6-bits	10-bits	6-bits	10-bits
Continuous Growth	54.6 dB	57.6 dB	41.9 dB	46.7 dB
Constant Truncation	30.5 dB	33.0 dB	25.9 dB	27.2 dB
Variable Truncation Scheme	54.5 dB	57.5 dB	41.9 dB	46.7 dB

5. DYNAMIC KERNEL FUNCTION FFT ARCHITECTURE

The hardware implementation and design considerations of the dynamic kernel function FFT with variable truncation scheme are discussed in this chapter. The top most view of a digital microwave receiver is presented, followed by a few design techniques applied on the architecture of the FFT. The internal and external hardware components of the FFT are also described.

5.1 Digital Microwave Receiver

The generic model of the proposed digital wideband receiver is presented in Fig. 5.1. The term "generic" simply implies that this flow chart is the base receiver model under study. Various other digital receiver designs may be derived from this model based on the same or subset of functionality but with a different requirements and specifications. Weather Doppler radars and sonars for mapping the ocean floor are prime examples of a receiver requiring multi-tone signal detection and tracking with different signal conditioning algorithms.

Referring to the base receiver flow chart illustrated in Fig. 5.1, the receiver first transforms the time domain data into frequency domain, with the intention to acquire the frequency information for the incoming RF signal. Based on the algorithms presented in this chapter, the receiver has the capability to supply data for post processing blocks to identify and track the input signals, estimate the frequency, and determine the presence of

multi-tone signals. The real-time processing of the sampled data in addition to the high sampling rate of 2.048 GHz allows the receiver to continuously monitor the frequency spectrum of interest while tracking the detected signals without a lapse in time domain coverage.

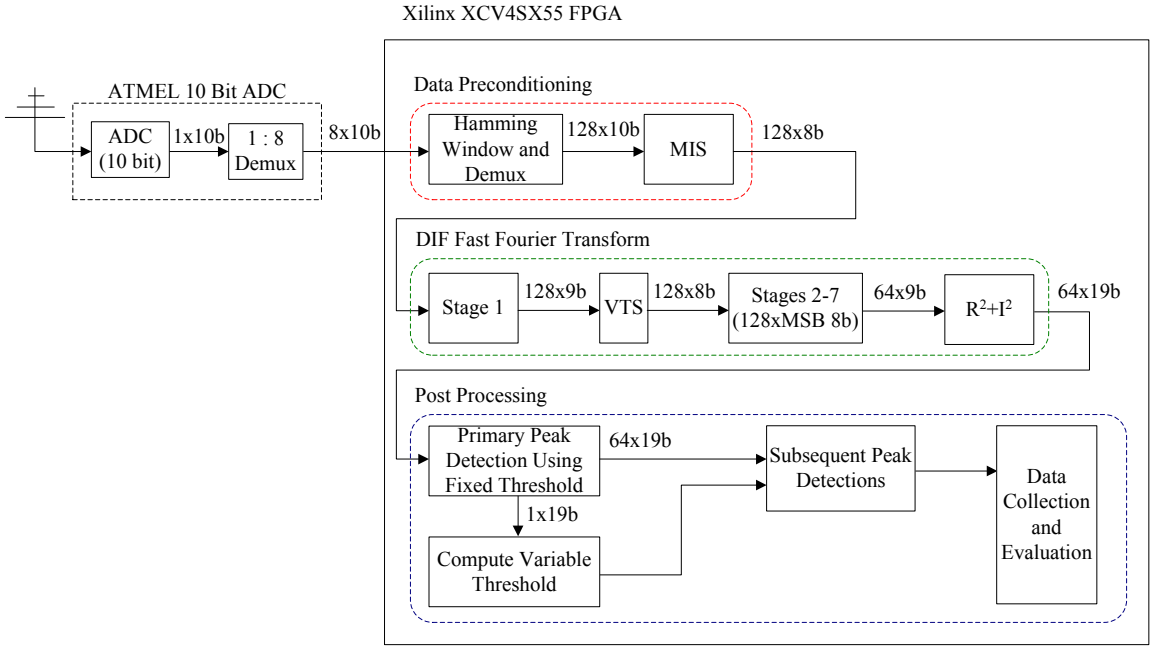

Fig. 5.1 Proposed Digital Receiver Flow Chart

The 10 bit digitized data sampled at 2.048 GHz first encounters the Hamming window function followed by the input demultiplexer (Demux), which acts as a data collector for acquiring the number of samples needed for the 128-point dynamic kernel function FFT. The result is a spectrum with 64 frequency bins, each separated by intervals of 16 MHz, for a full bandwidth of 1.024 GHz. The first and last three frequency bins are intentionally cleared to zero serving as a guard band for limiting the effects of the DC bias as well as aliasing of high frequency components when determining the frequency content of the incoming signal. Aliasing is a potential problem for the system since no external digital filters are designed to constrain the input

frequencies to a specific range. The guard bands limited the number of valid frequency bins from 64 to 58, indicating an effective bandwidth of 912 MHz. The FFT outputs a new set of transformed data at a rate of 16 MHz, or 62.5 ns.

The fixed detection threshold, *th*, is determined for signal detection based on the flag indicators from the FFT's variable truncation scheme in addition to the mean value for the sum of frequency bin magnitudes. A binary tree based peak detection block is used to determine the primary and secondary signals based on their descending order of magnitudes. A peak is located when the preceding and subsequent frequency bins have lower magnitudes than the current frequency bin. In the case where only the magnitude is greater than the detection threshold, this condition alone is not sufficient for defining the frequency bin as a signal. In the case when the input signal frequencies are not represented by the frequency bin, the input signals are estimated to the nearest frequency bins for the detected peaks.

5.2 Dynamic Kernel Function FFT

5.2.1 Architecture

The architecture for decimation-in-frequency is used for implementing the 128-point FFT. Based on architecture, the stages are divided into shorter DFTs as the data progresses through the FFT stages. The later stages reuses the shorter DFTs within the same stage, and thus it is folded and reused, as illustrated in Fig. 5.2 for a 4-point FFT. For effective hardware realization, the butterfly is reused for computation.

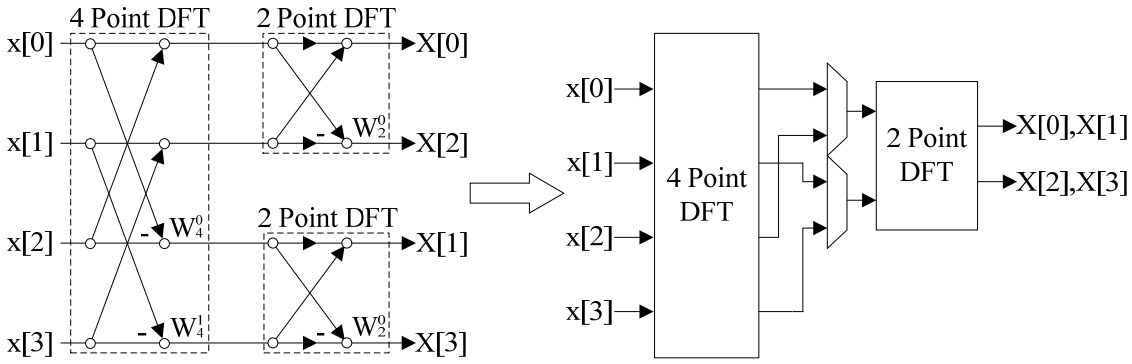

Fig. 5.2 4-point FFT with Folding

The folding technique can be extended for N-point FFT, minimizing hardware to form a more efficient architecture. A design challenge of applying this technique is that the data rates doubles for the folded stage in comparison with the previous stage. To accommodate the routing delays for the FFT, the folding technique is only applied to stages two, three, and four. The final architecture of the FFT is shown in Fig. 5.3, where there are a series of multiplexers (Mux) at the outputs of stages one, two, and three to accommodate data rates required for the folded stages (fold by 3).

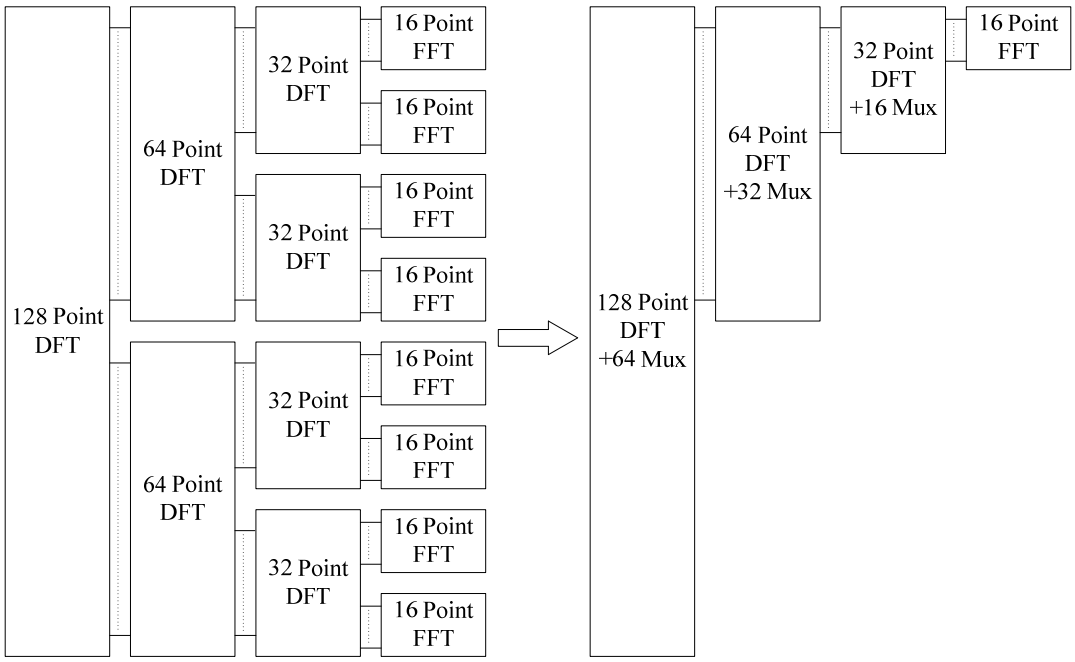

Fig. 5.3 128-Point Dynamic Kernel FFT with Folding

This results in a significant FPGA slice reduction, from 84 % to 32 % out of 24,576 slices available. Various FFT folding architectures were synthesized using Xilinx ISE 8.2i and the total hardware resources used is tabulated in Table 5.1.

Table 5.1 Synthesis Results

Architecture	Total Slices	Slices Occupied (%)	Equiv. Gate Count
Virtex 4 - SX55	24,576	-	-
No Folding	20,764	84	456,945
Folding by 1	12,334	50	261,570
Folding by 2	9,238	37	182,876
Folding by 3	7,916	32	156,524

5.2.2 Butterfly Design

The implementation of the butterfly is derived from the radix-2 architecture for general digital signal processing (DSP) chips [54], shown in Fig. 5.4.

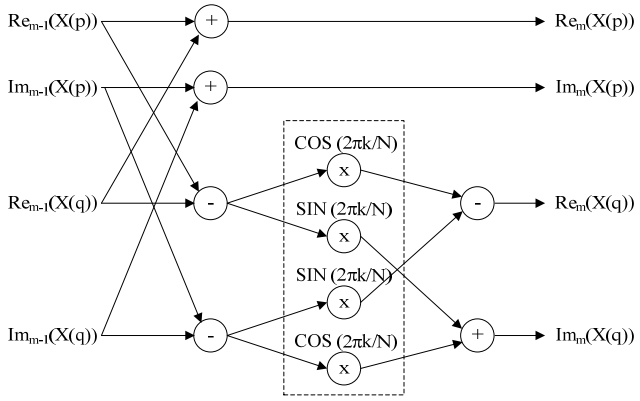

Fig. 5.4 Butterfly Implementation

Following the decimation-in-frequency FFT algorithm, the 8-bits real and imaginary components from the previous stage are added and subtracted accordingly before the twiddle factor is multiplied to the result. The dynamic kernel function is used for the cosine and sine representation of the twiddle factor. Shift and add (or subtract)

operation is used instead of general purpose multipliers shown in Fig. 5.4 for multiplication of sine and cosine components.

Careful considerations are taken to ensure numerical precision using the dynamic kernel function for shifting and adding, For instance, suppose the input to the dynamic kernel function is 4-bits with a value of 0111_b (7_d). The sine or cosine component of the dynamic kernel function intended is 7/8 for a unit circle expansion of eight. In the ideal case, the result is 7 x (7/8) = 49/8 or 6 and 1/8.

Using the shift and add implementation, the dynamic kernel function of 7/8 can be decomposed to (1/2) + (1/4) + (1/8) or 1 - (1/8). Fig. 5.5 shows the result when truncation takes place prior to addition/subtraction of the intermediate results.

$$\begin{array}{rcl|rcl} \frac{1}{8}(0111) & & 0000 & & & \\ \frac{1}{4}(0111) & = & 0001 & 1\,(0111) & = & 0111 \\ +\ \frac{1}{2}(0111) & & +0011 & -\ \frac{1}{8}(0111) & & -\ 0000 \\ \hline & & 0100 = 4 & & & 0111 = 7 \end{array}$$

Fig. 5.5 Trucation of Intermediate Results

The truncation method used for the implemented butterfly is displayed in Fig. 5.6, where the fractional precision of the intermediate results are kept until the final arithmetic calculation. This yields a result of six. In contrast with the method implemented in Fig. 5.5, the error is only 2 % compared with 35 % and 14 % when evaluated against the ideal result of 6.125.

$$\begin{array}{rcr} \frac{1}{8}(0111) & & 0000.111 \\ \frac{1}{4}(0111) & = & 0001.11 \\ +\ \frac{1}{2}(0111) & & +0011.1 \\ \hline & & 0110.001 \\ & & = 0110 = 6 \end{array} \qquad \begin{array}{rcr} 1\,(0111) & = & 0111 \\ -\ \frac{1}{8}(0111) & & -0000.111 \\ \hline & & 0110.001 \\ & & = 0110 = 6 \end{array}$$

Fig. 5.6 Butterfly Implementation

Note the truncation mentioned here is truncating away the fractional bits, thus setting a binary point of zero with a 9-bit output for the butterfly when implemented. In order to prevent overflow, the outputs for the second level of adders/subtractors in Fig. 5.4 are saturated when values exceeds the positive or negative extremes of the 9-bit 2's complement number.

5.2.3 Dynamic Kernel Function

The cosine and sine terms in Fig. 5.4 are implemented using the 6-bit dynamic kernel function mapping shown in Table 4.1. For the ideal twiddle function W_N^9, the 6-bit dynamic kernel function is (29/32,14/32) when scaled by 32. (29/32, 14/32) represents the cosine and sine values of the kernel function W_N^9 for hardware implementation. The design considerations needed to implement 29/32 are shown below.

First, 29/32 is decomposed into fractions with powers of two in the denominator, shown in Eq. (5.2.1). This will define the list of shifts needed to replace the multiplication.

$$\frac{29}{32} = \frac{16}{32} + \frac{8}{32} + \frac{4}{32} + \frac{1}{32} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} \tag{5.2.1}$$

An alternative expression is presented in Eq. (5.2.2)

$$\frac{29}{32} = \frac{32}{32} - \left(\frac{2}{32} + \frac{1}{32}\right) = 1 - \left(\frac{1}{16} + \frac{1}{32}\right) \tag{5.2.2}$$

Each equation will yield to different implementations with different hardware usage. Given a 9-bit input, A[8:0], to compute A[8:0]x(29/32), the computations needed are shown in Fig. 5.7, where B[8:0] is the final result after scaling back the unit circle.

$$\begin{array}{rl}
 & A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1.A_0 \\
 & \quad A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2.A_1\ A_0 \\
 & \quad\quad A_8\ A_7\ A_6\ A_5\ A_4\ A_3.A_2\ A_1\ A_0 \\
+ & \quad\quad\quad\quad A_8\ A_7\ A_6\ A_5.A_4\ A_3\ A_2\ A_1\ A_0 \\
\hline
 & B_8\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0
\end{array}$$

Fig. 5.7 Shift and Add Operation Corresponding to Eq. (5.2.1)

The first term is the input value divided by two, followed by divisions of four, eight, and thirty-two. The terms are added together and shifted back to yield a 9-bit value denoted by B. There are two ways to implement Fig. 5.7 in hardware. The first is shown in Fig. 5.8.

$$\begin{array}{rl}
 & A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1.A_0 \quad (Z) \\
 & \quad A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2.A_1\ A_0 \quad (Y) \\
 & \quad\quad A_8\ A_7\ A_6\ A_5\ A_4\ A_3.A_2\ A_1\ A_0 \quad (X) \\
+ & \quad\quad\quad A_8\ A_7\ A_6\ A_5.A_4\ A_3\ A_2\ A_1\ A_0 \\
\hline
 & B_8\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0
\end{array}$$

Fig. 5.8 First Implementation for Eq. (5.2.1)

The additions are performed in a linear faction. First, the third and fourth terms are added in the 10-bit adder X, where A_1 and A_0 are omitted prior to addition since they

do not contribute any precision changes in the computation. The LSB bit in the sum of adder X is truncated away prior to addition with the second term in 10-bit adder Y. The LSB bit in the sum of Y is truncated away and added with the full 9-bits of the first term using adder Z, the last two bits from the sum of adder Z are truncated away after additions since the impact of the fractional binary values are already considered and the binary output B needs to return to an integer value. The hardware is illustrated in Fig. 5.9, where three 9-bit adders are cascaded.

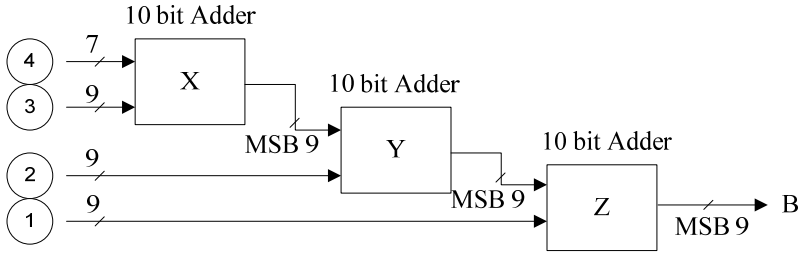

Fig. 5.9 Hardware for First Implementation of Eq. (5.2.1)

The second alternative design to Eq. (5.2.1) is presented in Fig. 5.10.

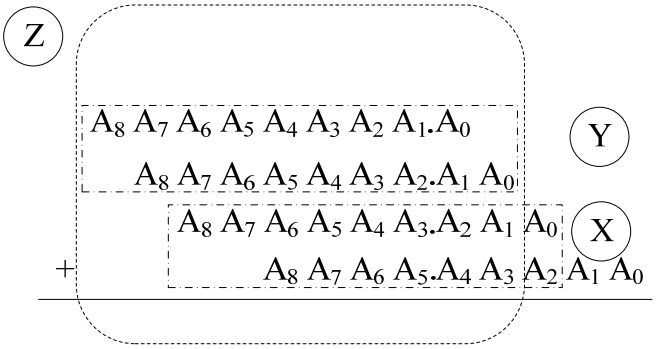

Fig. 5.10 Second Implementation for Eq. (5.2.1)

Similar to the first implementation, 10-bit adder X is used to add the third and fourth terms. However, this time, an 11-bit adder Y is used to sum the first two terms in parallel with adder X. A zero is padded to the end of the first term in adder Y to preserve the precision. Finally, the 11-bit adder Z is used to sum the results of adder X and Y.

Adder Z is 11-bits instead of 12-bits because adder Y already included an extra bit in the output of the sum to prevent overflow. The hardware implementation is illustrated in Fig. 5.11.

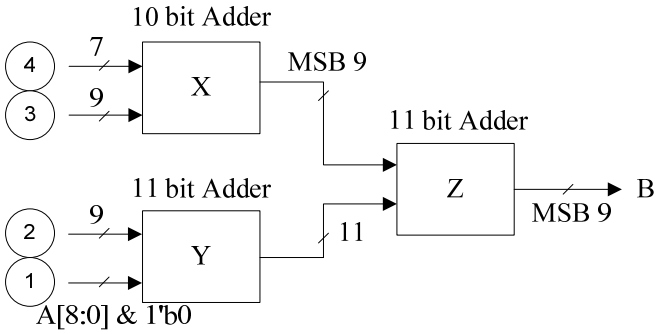

Fig. 5.11 Hardware for Second Implementation of Eq. (5.2.1)

Eq. (5.2.2) utilizes a subtraction operations and the implementation is demonstrated in Fig. 5.12.

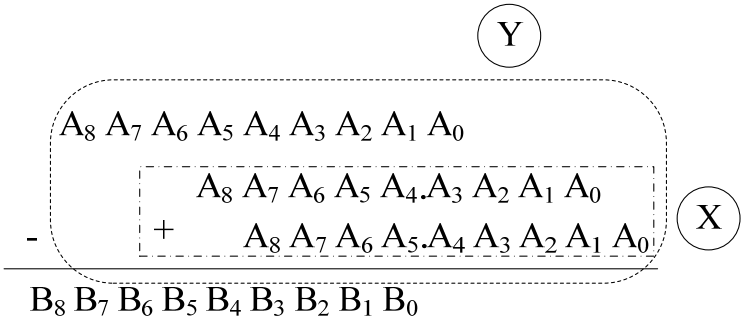

Fig. 5.12 Implementation for Eq. (5.2.2)

An 11-bit adder X is needed to compute the second term padded with a zero with the third term. The first term is then padded with four zeroes, and the sum of X is subtracted from the padded number using a 14-bit subtraction. The MSB 9 bits are taken from the result. The hardware implementation is presented in Fig. 5.13.

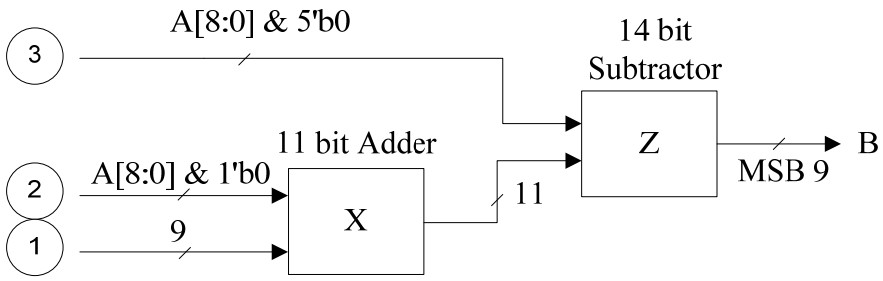

Fig. 5.13 Hardware Implementation for Eq. (5.2.2)

Each of the three implementations has its own advantages and drawbacks. Suppose the adders are modeled as ripple carry adders using full adders (FA). Then the amount of hardware used will be the amount of full adders need. The delay of the datapath may also be estimated based on the number of full adders passed. The results are tabulated in Table 5.2.

Table 5.2 Hardware Design Cost and Delay

Figure Number	Description	Total Hardware (FA)	Delay
Fig. 5.9	Cascaded Adders	30	30
Fig. 5.11	Adders in Parallel	32	22
Fig. 5.14	Subtraction	25	25

Cascaded adder design is the simplest to implement, but with the highest delay. Using adders in parallel will yield the fastest running design, at the expense of additional full adders. The dynamic kernel function design using subtraction utilizes least amount of hardware with decent delay characteristics.

5.2.4 Variable Truncation Scheme

Three steps are required for variable truncation. First, a comparator is used to generate a status flag. Second, the status flag are accumulated to make a final decision on

which bits to take. Finally, a multiplexor is used to select the appropriate bits based on the accumulated results.

For the implemented FFT, the inputs of each FFT stages are limited to 8-bits. After the butterfly operation, each stage will have a 9-bit 2's complement output prior to truncation. A comparator is needed to determine whether to take the MSB or LSB 8-bits. This determination is made based on the two MSB bits of the 9-bit number. If these bits are both 0 or 1, the most significant bit is not needed since the number may be represented by the LSB 8 bits. This is illustrated in Fig. 5.14 for testing a 9-bit input A[8:0].

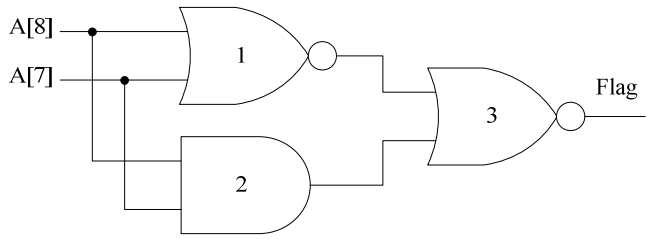

Fig. 5.14 Comparator for Variable Truncation Scheme

The outcomes of these comparisons are summed together using logical OR gates organized in a tree to quickly yield an overall status flag to represent all the comparator results. A logical one in the output of the above circuit indicates that at least one of the 9-bit outputs exceeds 128, signifying the number cannot be fully represented by an 8-bit 2's complement number. The only option is to take the MSB 8-bits for all the outputs of this particular FFT stage, which effectively truncates the LSB bit away.

A logical zero indicates the next stage will use the LSB 8-bits, truncating the MSB bits away. Since the status flags ensure that no values from the previous stage's 9-bit 2's complement outputs exceed the value 128. This guarantees that the MSB bit truncated away has no valuable information.

The variable truncation scheme utilized here compensates what traditional fixed-precision FFT lacks in preserving valuable least significant bits when processing a very weak input signal. Due to the nature of the folding FFT architecture, the operating frequency increases as nested folding is applied. This will result in range of points processed at different points of time, as illustrated in Fig. 5.15.

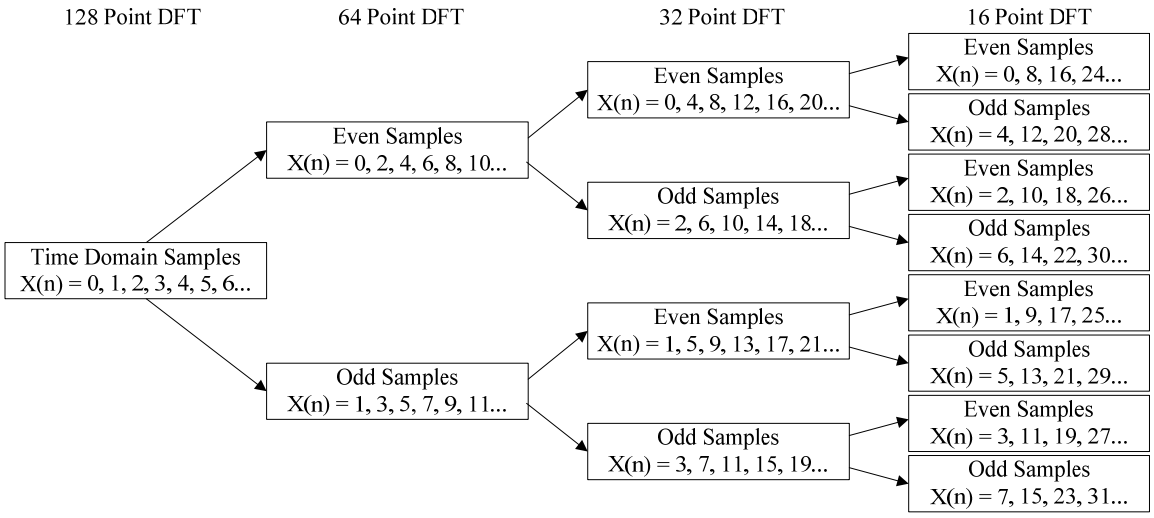

Fig. 5.15 Time Samples Processing in FFT Folding Architecture

Since the range of data points processed is different at any given point of time, the variable truncation scheme will present a problem when used within a nested folding architecture. The status flags utilized to determine the bit range used for the next stage will only be based on a subset of the data points processed. The truncation determination after the 128-point DFT will be precise since the comparison takes place between all the data points. The problem starts to occur in stage 2 for the 64-point DFTs. The FFT will complete the calculation for the even samples before the odd samples are evaluated,

indicating that the status flags used for variable truncation will be based on a local optima for the next stage.

The outcome from this effect is different scaling used for the sub ranges of data points when performing the DFT calculation in stages two and three. Due to the above constraint imposed by the folding architecture, variable truncation is utilized at the output of stage 1 only. Thus the worst case performance loss for weak signal detection is decreased by a factor of six (6 out 7 stages truncates away 1 bit) and translates to approximately 0 to 18 dB performance lost in the two different sub-ranges of the final FFT dynamic range. However, the above analysis assumes the worst case condition where the presence of spikes only appears in certain sub-ranges of the sampled data.

5.3 Data Preconditioning

Following Fig. 5.1, this section contains discussions on the window function and an extension to variable truncation scheme for selecting various 8-bit input combinations.

5.3.1 Window Function

Due to the limitation on the size of the FFT, the sampled time domain data is also limited for a finite duration. This simple limitation on data collection causes an implied rectangular window applied to the sampled input with unit weight. This leads to spectral leakage when a non-integer number of periodic time domain data is sampled at the input, where the end points of the observation window are disjoint. Spectral leakage is when the magnitudes of the frequency bins is non-zero for bins not related to the input signal frequency. This presents a problem for a weak input signal, where a peak and side-lobe of

the input signal cannot be clearly distinguished. Spectral leakage also occurs when the input signal frequency is not represented defined or near the frequency values represented by the FFT bins.

The application of window function with various weights for each time domain sample increases the instantaneous dynamic range of multi-tone signals by attenuating the side-lobes of the input signals. Unfortunately, applying a window will widen the main-lobe of the signal, which indicates that the frequency bin difference between two signals may not be too close due to aliasing in the power spectrum. The evaluation of windowing function is limited to rectangular, Hanning, and Hamming windows. May digital signal processing text may be referenced for more detailed discussion on window functions. The comparisons of the three different functions are tabulated in Table 5.3.

Table 5.3 Comparison of Different Window Function Properties (Fs = 2.048 GHz)

Type of Window (n=128)	Main-lobe Width (-3 dB)	Relative Sideband Attenuation	Leakage Percentage
Rectangular	14 MHz	-13.3 dB	9.14 %
Hanning	22 MHz	-31.5 dB	0.05 %
Hamming	20 MHz	-42.6 dB	0.03 %

Hamming window is selected for its superior performance in sideband attenuation and limiting the leakage power in the frequency spectrum. The sideband attenuation is fairly close to the performance expectation of an 8-bit ADC, where the SNR is 48 in the ideal case.

The implementation of the Hamming window function is similar to the dynamic kernel function, where the value for each coefficient will be scaled by powers of two and represented as fractional number with respect to the scaled binary value. The sampled

time domain data is multiplied with the window coefficients using dedicated 18 bit multipliers available on the Virtex-4 FPGA. For the Xilinx Virtex-4 SX55 FPGA model, 512 high speed dedicated multipliers are available and does not count towards the total number of slices available for implementing digital logic. The general architecture for the window function implementation is shown in Fig. 5.16.

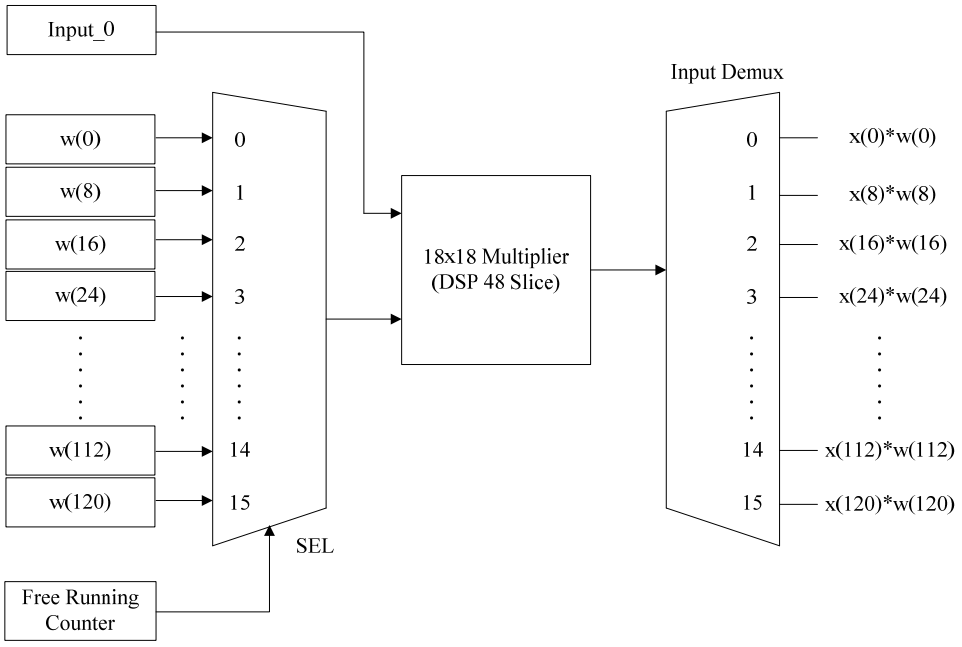

Fig. 5.16 Window Function Implementation (Cell 0)

Sixteen window coefficients are pre-stored in each cell of the window function implementation, resulting in a total eight cells required for processing all 128 time-domain samples. The 16 window coefficients are selected based on the output of a 4-bit free running counter, where the count starts from 0 to 15 and overflows back to zero after maximum value of 15 is reached. The outputs of the window function are collected via an input demux to the dynamic kernel FFT. Sixteen values from each of the eight windowing cells are collected and fed to the 128-point FFT at the same time.

In order to satisfy the real-time requirements of the ADC3255 board, the eight demuxed time-domain samples from the 10-bit Atmel ADC must be processed in parallel with the above implementation at every 256 MHz. The determination for the storage of the window coefficients follows the corresponding time domain sample data order, n, indicated in Table 5.4 for each window function implementation cell. The design is flexible and may be adapted for other types of window functions, such as Hanning, Blackman, and Kaiser windows [49], etc.

Table 5.4 Hardware Responsible for Applying Hamming Window Function

Windowing Hardware	Time Domain Input Sample x(n) (n=0,1,2…..127)
Cell_0	0, 08, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 096, 104, 112, 120
Cell_1	1, 09, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 097, 105, 113, 121
Cell_2	2, 10, 18, 26, 34, 42, 50, 58, 66, 74, 82, 90, 098, 106, 114, 122
Cell_3	3, 11, 19, 27, 35, 43, 51, 59, 67, 75, 83, 91, 099, 107, 115, 123
Cell_4	4, 12, 20, 28, 36, 44, 52, 60, 68, 76, 84, 92, 100, 108, 116, 124
Cell_5	5, 13, 21, 29, 37, 45, 53, 61, 69, 77, 85, 93, 101, 109, 117, 125
Cell_6	6, 14, 22, 30, 38, 46, 54, 62, 70, 78, 86, 94, 102, 110, 118, 126
Cell_7	7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95, 103, 111, 119, 127

5.3.2 Multiple N-bit Input Selections (MIS)

The methodology used is the same as variable truncation scheme. It is named differently to distinguish between the variable truncation taking place in between the FFT stages versus the input bit selection. The only different in this case, is that MIS will attempt to scale the weak signal up to the full 8-bits to maximize the 8-bit input usage of the FFT. Given a 10-bit sampled data, A[9:0], the input bit selection are listed in Table 5.5.

Table 5.5 Multiple N-bit Input Selections (N=10)

Status Flags								Bit Selection
x_1	x_2	x_3	x_4	x_5	x_6	x_7	X_8	
1	1	1	1	1	1	1	1	A [9:2]
0	1	1	1	1	1	1	1	A [8:1]
0	0	1	1	1	1	1	1	A [7:0]
0	0	0	1	1	1	1	1	A [6:0] & 1'b0
0	0	0	0	1	1	1	1	A [5:0] & 2'b00
0	0	0	0	0	1	1	1	A [4:0] & 3'b000
0	0	0	0	0	0	1	1	A [3:0] & 4'b0000
0	0	0	0	0	0	0	1	A [2:0] & 5'b0000

Out of the 10-bit input, the range of bit selection ranges from the MSB 8 bits to LSB 3 bits with 5 zeros padded. Instead of a 1-bit status flag, at least 7 status flags are needed. The status flag combination for the last row may be omitted as a don't care since the previous status flag will determine whether to take the LSB 4 bits or 3 bits. The comparator needed for the input level selection is similar to Fig. 5.14 and it is re-illustrated in Fig. 5.17.

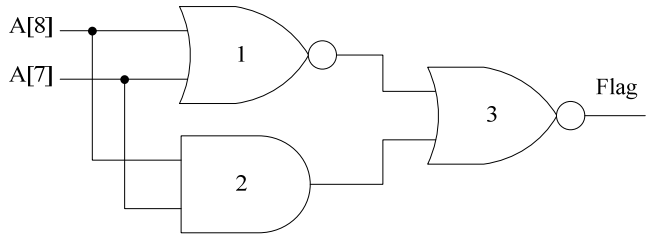

Fig. 5.17 Comparator for Variable Truncation Scheme

The only design difference would be the number of MSB bits to consider. For instance, to determine whether bits A[7:0] will contain all the information of the sampled waveform, the first three MSB bits must be checked for all zeros or ones. In Fig. 5.17, gates 1 and 2 will become 3 input gates while the remaining connections stay the same.

Seven different comparator designs are need for each 10-bit sampled data to determine eight different 8-bit input utilizations.

For a 128-point FFT after data collection, there will be 128 sampled data. Each of these will have seven status flag to indicate which eight bits to take. The status flag remained as a 7-bit representation instead of mapping it to a 3-bit number. If mapped as a 3-bit number with values between 0 and 7, a complex status flag accumulation network will need to be designed. Leaving the status flag as a 7-bit number, the or network will just be 7 time bigger when compared with variable truncation scheme for choosing 8 out of 9 bits. Each status flag, x_1 to x_7, are accumulated independently. Once the status flags are accumulated for all 128 sampled data, the 7-bit status flag is encoded to a 3-bit number via a priority encoder with the MSB 8-bits having the highest priority. Finally, a 8-to-1 mux is used at the outputs of the MIS block to select the same 8-bit combinations for all sampled data.

5.4 Post Data Processing

5.4.1 Peak Detection

Peak detection is critical in determining a given FFT bin as a candidate for possible signal detection. The peak detection block is implemented with a binary tree based comparisons between the frequency bin magnitudes as shown in Fig. 5.18.

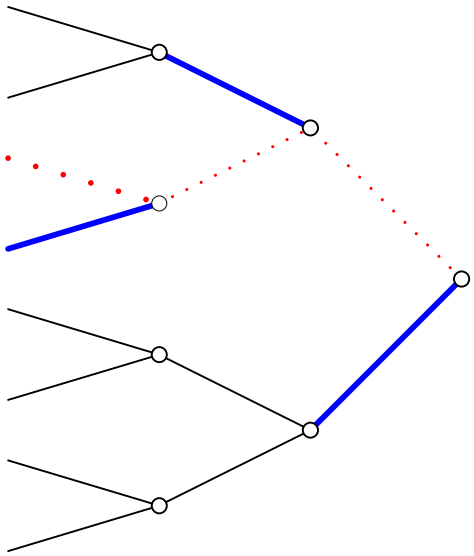

Fig. 5.18 Binary Tree Based Comparison

The computational complexity O(n) of the binary tree based search is log n for each peak signal. The tree may be reused for finding subsequent peak magnitudes. Thus for a four-tone signal detector, O(n) = 4 log n. Once the peak detection block finds the highest magnitude in the current spectrum, the bin magnitudes for the current search peak and its neighbors must be cleared or set to zero to avoid the current main-lobe magnitudes interfere with the next search for the peak. The process is iterative for interested number of signal detection. For each node in the binary tree comparison logic, the magnitude of the higher value and an 1-bit status flag are passed to the next node. The next node of higher hierarchy performs the same comparison logic and appends the new status flag in front of the existing status flag. The final status flag will indicate the precise bin location of the highest magnitude in the current spectrum.

The advantage of this design is the ability for the circuit to avoid the detection of a local maxima. Suppose in Fig. 5.18, the bin with the highest magnitude is the red dotted line. Then any of the bolded blue lines in the figure are candidates for the next highest

value in the spectrum. If one assumes that the final comparator before the output is the next highest magnitude in the spectrum, then the solution for the second highest signal would be a local maxima.

5.4.2 Dual Thresholding

The detected peaks are compared with the threshold values before signal detection is justified. Typical designs operate on one threshold, adaptive or fixed, where peaks detected above the threshold are considered as a signal.

Without any input stimulus, the average noise floor from the output of 128-point dynamic kernel function FFT is about 2 dB prior to normalization. The lower threshold is placed at 14 dB above the average noise floor value. This is a fixed threshold placed at 16 dB above zero in the frequency spectrum.

Two different upper threshold settings are studied in the preliminary analysis of applying two thresholds, upper and lower conditions for signal detection. The upper thresholds are set at 15 and 18 dB below the magnitude of the primary signal. Note that only one upper threshold is used per implementation of the digital receiver. Since the magnitude of the primary signal may be different depending on the amplitude and signal environment, the upper threshold will also vary accordingly while maintaining the 15 or 18 dB difference.

The inspiration for setting a second threshold is to reduce the hardware consumption related with adaptive thresholding techniques while minimizing the false alarm rate. Techniques for improving the total dynamic range of receivers such as automatic gain control and variable truncation scheme are susceptible to varying noise

floor magnitudes, which leads to false alarms when only one fixed threshold is used. Instead of computing the mean and variance in the current frequency spectrum for adaptive threshold determination, a fixed threshold, *Th1* in Fig. 5.19, can be set from the magnitude of the primary signal. This threshold should also account for FFT arithmetic truncation and ADC quantization loss.

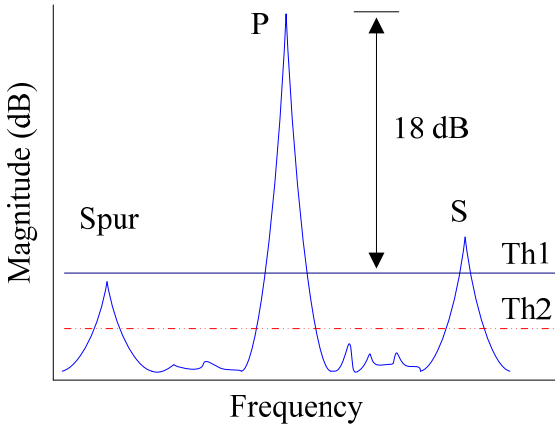

Fig. 5.19 Dual Thresholding

The decibel value used for defining the upper threshold expresses the upper threshold relative to the primary magnitude. This implies a division is required to find the exact placement of Th1 for every frequency spectrum observation.

Recall that the FFT output is $R^2 + I^2$. For the example shown in Fig. 5.19, a conversion is required to find the ratio between the primary magnitude P and *Th1* in numerical scale.

$$10\log_{10}\left(\frac{P_{mag}}{Th_1}\right) = 18dB \tag{5.4.1}$$

$$\frac{P_{mag}}{Th_1} = 10^{18/10} = 63 \approx 64 \tag{5.4.2}$$

Taking ten to the (18/10) power, the ratio is approximately 64. This is convenient in hardware implementation using 2's complement representation. Instead of designing a complex circuit for division, the primary magnitude in binary form only requires six right shifts.

$$Th_1 = P_{mag} / 64 \tag{5.4.3}$$

For setting the upper threshold as 3, 6, 9, 12, 15, and 18 dB, the primary magnitude in absolute scale expressed in 2's complement format is right shifted 1, 2, 3, 4, 5, and 6 times for the approximate value of Th1.

Dual thresholding is simple to implement without excessive hardware resource requirement while improving the second signal false alarm rate. This benefits hardware constrained FPGA based digital receivers for signal acquisition and detection.

The extra threshold minimizes the effects of quantization and truncation on second signal false alarms. For signal detection, the secondary magnitude must be greater than both upper and lower thresholds. The process may be repeated for multiple signal detection based on the upper threshold set by the primary signal. The only drawback applying this technique is the trade off between a smaller constrained IDR versus second signal false alarm rate improvement.

6. DYNAMIC DESIGN FLOW AND PROTOTYPING HARDWARE

This chapter introduces the tools, procedures, and hardware used to implement the proposed dynamic kernel function FFT algorithms on the FPGA. The digital receiver is mapped on the FPGA with externally connected signal generators to supply the analog and clock sources required.

6.1 Design Flow

Matlab and Xilinx toolset are used extensively to meet the design and timing requirements needed for the digital receiver proposed. The top level view is illustrated below.

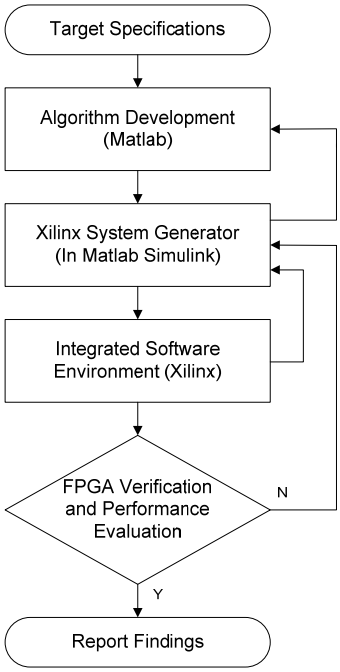

Fig. 6.1 General Design Flow (Top-Most Level)

Fig. 6.1 illustrates a very general overview of the design process. Matlab is used to simulate the algorithms intended using fixed-point format. The digital receiver hardware is designed and simulated using Xilinx System Generator as shown in Fig. 6.2. This is component-based design environment within the Matlab Simulink.

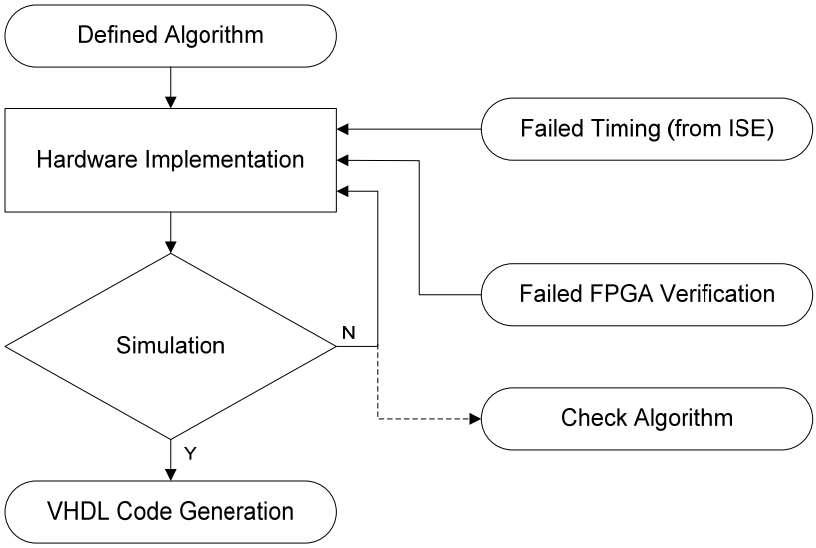

Fig. 6.2 System Generator Design Flow

A failed mapping and timing report from Xilinx Integrated Software Environment (ISE) will require a design optimization to conserve hardware resources and insert pipeline stages. A successful simulation in System Generator does not guarantee the generated VHDL code will run accordingly on the FPGA. This is true when dividing the overall design into separate blocks for generation. Even in the case when the design is generated in whole, the design may still be composed of blocks driven by different clock rates. The FPGA design may still fall short without proper register insertion in certain areas between multi-rate blocks.

The Xilinx ISE software performs a series of linear processes as demonstrated in Fig. 6.3. The user needs to manually integrate the generated VHDL code from system

generator with debugging cores from Chipscope, multi-rate clock generation from digital clock manager, and setup the input/output ports with the physical FPGA pins. The last item is included in the design kit supplied by the board manufacturer. The design kit also includes controls to adjust the ADC gain, PCI and DRAM data management, JTAG control signals for boundary scan testing, as well as internal or external clock selection.

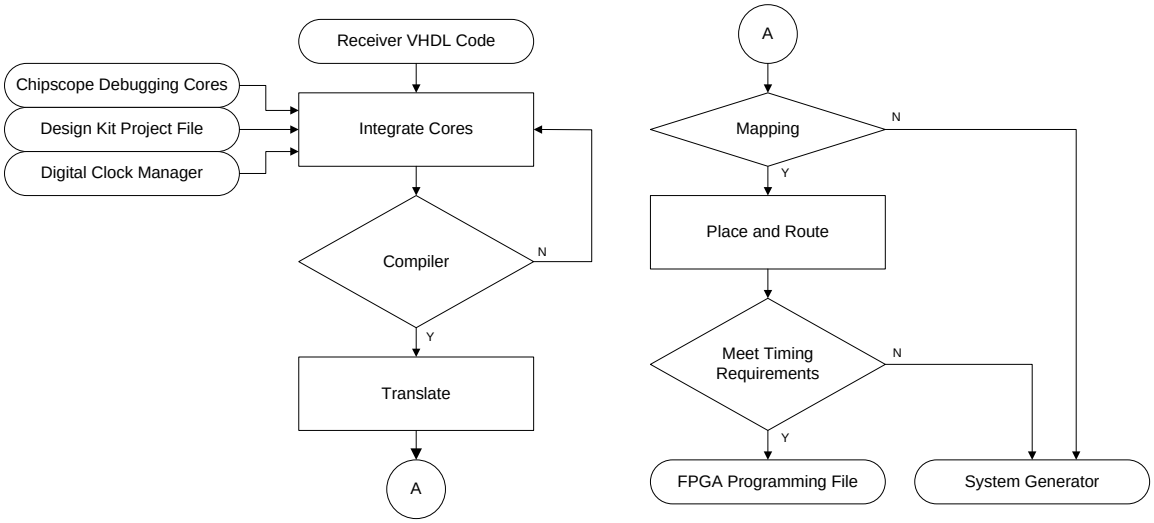

Fig. 6.3 Xilinx ISE Design Flow

The compiler handles general syntax checking and tries to optimize the system generator design. The optimization is not perfect for sequential designs, and the user is still mostly responsible to minimizing resource usage and datapath depth between registers. The translate process merges the incoming VHDL netlist with the appropriate Xilinx IP cores. The designs are then mapped on the available target FPGA resources. If the design is too big, this is where the program stops and it will generate an error. Mapping at this point only approximate the amount of resource the current design may need. The place and route operation then attempts to satisfy the timing constraints requested by the user. For a design with high data throughput rate, the user needs to make

constant changes to the architecture to ensure timing is met. Once place and route passes without any errors, ISE will generate a bitstream programming file for the FPGA. Xilinx's Chipscope Analyzer is used to verify the hardware design.

6.2 Prototyping Board

The medium for implementing the prototype digital receiver design is the Delphi ADC3255 PCI Mezzanine card (PMC), which contains a 10 bit Atmel ADC with Xilinx Virtex 4 FPGA. The general structure of the board is shown in Fig. 6.4.

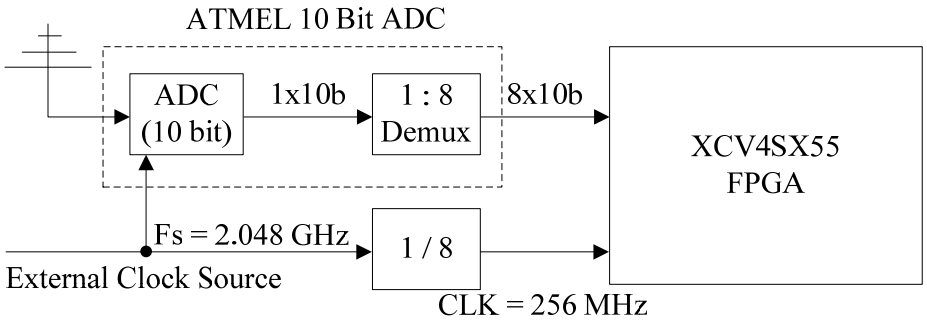

Fig. 6.4 ADC3255 Architecture

The clock is sourced from an external signal generator with a sampling frequency of 2.048 GHz. The 10 bit Atmel ADC samples the data at 2.048 GHz and outputs eight 10 bit data every 256 MHz due to an in-built demultiplexor. This is also why the default system clock of the FPGA is one-eighth of external clock frequency at 256 MHz.

6.3 Test Bed Setup

The precise equipment setup and port connections are illustrated in Fig. 6.5. The analog sinusoidal inputs are sourced from Agilent N9310A RF signal generators, where the expected harmonic distortions are less than -30 dBc [55]. The dual-tone signal is

produced through a RF power combiner (series PD1120) from Instock Wireless Components.

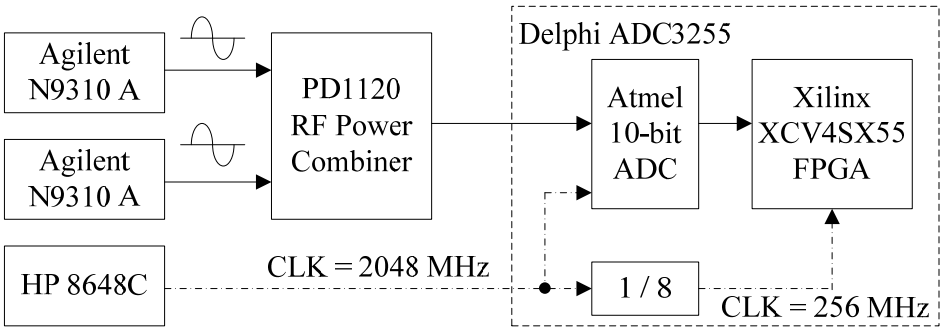

Fig. 6.5 Receiver Test Bed Setup

The output of the combiner is connected to the analog source port of the Delphi ADC3255 board, where the data is sampled at 2.048 GHz. HP 8648C signal generator provides the clock frequency at 0 dBm and the reported phase noise specification is -106 dBc/Hz [56]. The sampling frequency of 2.048 GHz is chosen to yield a FFT bin interval of exactly 16 MHz for conveniences in signal generation and analysis.

7. HARDWARE SYNTHESIS AND VERIFICATION

The number of slices available on the Virtex-4 is not sufficient for the implementing all the algorithms described, thus two separate 128-point dynamic kernel function FFT are designed, simulated, and tested on the FPGA. One design utilizes variable truncation scheme (VTS) after stage 1 with 8-bit FFT input taking either the MSB or LSB 8-bits from the ADC. The other design utilizes Multiple 8-bit Input Selections (MIS) for the 10-bit sampled output without VTS. Variable truncation is taken out of the second design to accommodate MIS. Single and dual-tone inputs are with various signal amplitudes

7.1 Hardware Synthesis

7.1.1 Xilinx FPGA Synthesis

The amount of hardware resources utilized is defined by the number of slices occupied in the VIrtex-4 SX 55 FPGA. Each slice in Virtex-4 series FPGA contains two 4-input look-up tables (LUT) and four 1-bit registers. In some cases, some slices may be used as a route-through, where no logical function is performed and the slice is used to improve timing. The design kit occupies about 8% of the available slices and it is included in the synthesis report below. The reported slice usage included additional

pipeline register inserted to meet the timing requirements of processing all the data inputs sampled at 2.048 GHz.

The components contained in the first design are illustrated in Fig. 7.1.

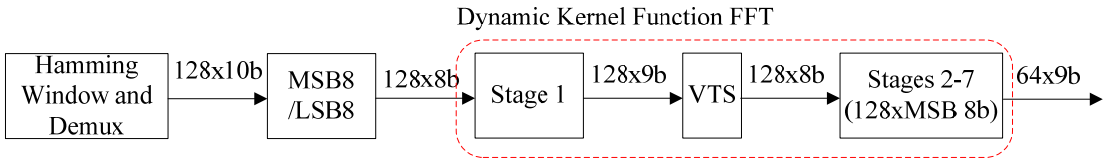

Fig. 7.1 Design 1 with Variable Truncation Scheme

In Fig. 7.1, the MSB or LSB 8-bits are selected from the 10-bit sampled input after the application of window function. Variable truncation is used after stage 1 of the FFT. The output of the FFT passes through the peak detection block to obtain the two highest signals in the spectrum with dual thresholding. The frequency and magnitudes of the primary and secondary signals detected are stored for 512 contiguous tests using Chipscope cores. The design uses about 89% of FPGA slices (21,917/24,576) with 45% of registers used (22,549/49,152). The total amount of 4-input LUT used is 55% (27,170/49,152), where 26,198 are used as logic, 199 as route-through, and 773 as shift registers.

The block diagram of the second design is shown in Fig. 7.2.

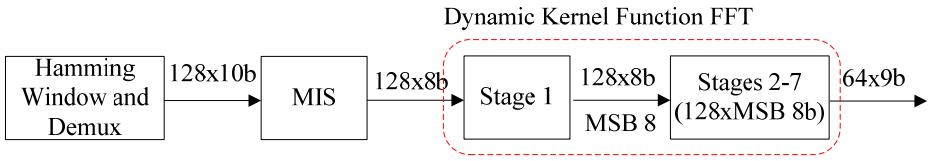

Fig. 7.2 Design 2 with Multiple 8-bit Input Selections

The second design does not have any variable truncation within the FFT, thus resulting in one LSB bit truncated away at the outputs of all stages. MIS contains the logic required to select eight different levels of 8-bit data from the 10-bit sampled input.

Variable truncation scheme is removed from the output of stage 1 to implement MIS. The outputs of the FFT are fed to the same blocks described in design 1. The design uses about 94% of FPGA slices (23,269/24,576) with 44% of registers used (21,965/49,152). The total amount of 4-input LUT used is 61% (29,997/49,152), where 28,988 are used as logic, 234 as route-through, and 775 as shift registers.

The first design uses variable truncation scheme on the real and imaginary results of the first stage. The first stage has 128 real and 0 imaginary inputs. After the butterfly, typically there will be a total of 4 real and imaginary outputs combined. However, since the imaginary portion of the input is zero, for the first stage, there are a total of 64x3 9-bit outputs. Then variable truncation scheme is performed on these 192 outputs. Compared with the second design, where MIS is performed on only 128 10-bit inputs, the design uses 5% more FPGA resources in the second case due to the size of the comparators.

7.1.2 Projections for Future Designs

Given the synthesis report from the last section, suppose the user wants to maintain the current input and inter-stage bit precision of the FFT. If a change is required to use a 10-bit dynamic kernel to replace the 6-bit version implemented for this work, the follow would change in the dynamic kernel function designs.

In the optimal design of 6-bit kernel function, at least two arithmetic functions are needed to add or subtract the shifted operands in the worst case situation, where three out of six bits are ones. The design is illustrated in Fig. 7.3. If more than three bits are one, then subtraction operations may be used. Tabulating the number bit-wise connections and

full adders needed in this design, at least 48 bit-wise connections and 21 full adders are needed.

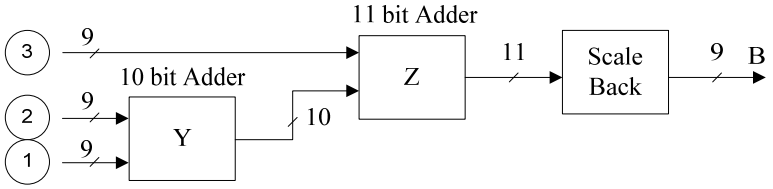

Fig. 7.3 Worst Case Design for 6-bit Dynamic Kernel

In the optimized design of 10-bit kernel function, at least four arithmetic functions are needed to add or subtract the five shifted operands in the worst case situation. The design is illustrated in Fig. 7.4. At least 108 bit-wise connections and 43 full adders are needed. The transition from 6-bit to 10-bit kernel would require a shade over 100 % of increase in the adders and bit-wise connections alone. The increase in hardware assumes that the input and inter-stage precisions are still maintained at 8-bits.

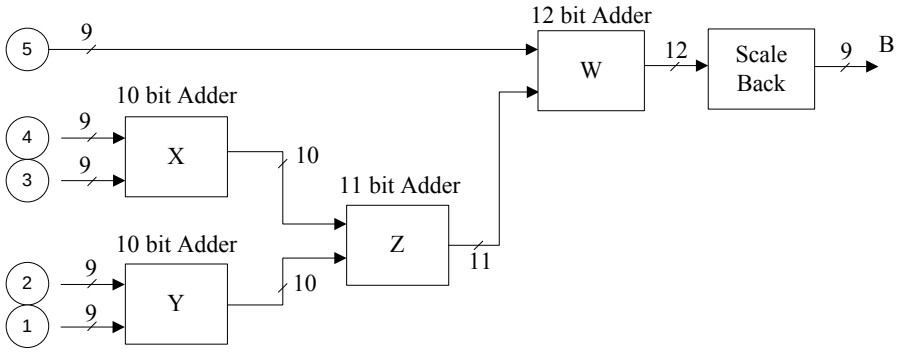

Fig. 7.4 Worst Case Design for 10-bit Dynamic Kernel

Suppose designs with a higher inter-stage bit precision of 14-bits are required to improve the IDR of multi-tone signal detection. This is a 6 bit increase from the original 8-bit nets design. All the buses composed of real and imaginary signals in addition to adders used will be increased by 6 bits accordingly. Assuming the 6-bit dynamic kernel

function is not changed, this implies a 75 % increase in the dynamic kernel function FFT alone.

7.2 Simulated Performance Evaluation

The 128-point dynamic kernel function FFT was designed and simulated using the Xilinx System Generator for DSP version 8.2.02. Strong and weak single-signal detection performances are performed in System Generator and on the FPGA. Designs 1 and 2 discussed in 7.1 are simulated separately.

The sampling frequency is 2.048 GHz with random phased inputs. The frequency resolution is 16 MHz with 58 of the 64 bins tested. Additionally, 57 inputs with frequency values not directly represented by the frequency bins are also tested. The input frequency is swept from 48 to 960 MHz with 8 MHz intervals for a total of 115 test cases per input amplitude. The averaged primary signal magnitude, highest noise spur, and SFDR values are recorded in dB.

The ideal 10-bit ADC used for the simulations in section 7.2 to measure the impact of the dynamic kernel on the performance of the FFT. The full scale amplitude of a 10-bit 2's complement value is 511 to avoid saturation in the sampled data. The amplitude is decreased progressively to fit the full scale value of the LSB 9-bits, 8-bits, 7-bits, till 1-bit.

7.2.1 Variable Truncation Scheme (Design 1)

The single-signal simulation results for design 1 with no window, ideal Hamming window, and hardware implementation of Hamming window are tabulated in Table. 7.1.

The addition of the variable truncation blocks extends the maximum single-signal SFDR performance till amplitude of 63 (full scale value of LSB 7-bits).

Table 7.1 VTS Single Signal Performance (Design 1)

Design	Input Amp	Primary Signal	Noise Spur	SFDR
VTS1 (No Window)	511	42.32	11.73	31.86
	255	42.30	11.73	31.85
	127	42.25	11.61	32.17
	63	42.12	12.38	31.09
	31	35.96	11.07	26.40
	15	29.48	11.08	19.98
	7	22.40	10.94	13.12
	6	20.68	10.79	11.93
VTS1 (Ideal Window)	511	37.53	10.88	26.96
	255	36.85	11.61	25.59
	127	37.41	10.96	26.85
	63	36.69	11.87	25.19
	31	30.35	11.17	19.70
	15	23.86	11.59	12.83
	9	18.76	11.02	8.12
VTS1 (HW Window)	511	37.33	11.26	26.45
	255	36.92	11.83	25.55
	127	37.11	10.90	26.44
	63	36.77	11.73	25.41
	31	30.64	11.02	19.99
	15	24.42	11.25	13.57
	13	23.05	11.42	11.99

The weakest detectable signal with 100 % detection in all frequency values using the implemented hardware window is 31.8 dB down from the 10-bit full scale value of 511. This is a drop of 6.8 dB and 3.2 dB in dynamic range when no window and ideal windows are used prior to digitizing the data. Utilizing the dual-thresholding scheme, the expected dynamic range is 31.8 dB without any false alarms, but not all signals will be detected when the signals are pushed below this range.

7.2.2 Multiple 8-bit Input Selections (Design 2)

The single-signal simulation results for design 2 with no window, ideal Hamming window, and hardware implementation of Hamming window are tabulated in Table. 7.2.

Table 7.2 MIS Single Signal Performance (Design 2)

Design	Input Amp	Primary Signal	Noise Spur	SFDR
MIS (No Window)	511	42.33	11.47	32.22
	255	42.25	11.65	32.22
	127	42.27	11.39	32.37
	63	42.14	11.87	31.58
	31	41.88	12.08	32.13
	15	41.41	12.32	30.51
	7	40.36	14.45	28.18
	6	39.13	13.71	29.42
	3	37.96	18.25	23.37
	2	33.42	17.72	16.96
	1	25.24	18.90	11.44
MIS (Ideal Window)	511	36.90	11.21	26.16
	255	36.90	11.13	26.11
	127	36.78	10.43	26.93
	63	36.66	10.90	26.15
	31	36.34	11.11	25.53
	15	35.69	11.81	24.16
	9	30.91	11.91	19.34
	7	34.35	13.59	21.15
	3	31.18	14.49	17.01
	2	26.37	15.06	12.13
	1	19.18	14.33	5.25
MIS (HW Window)	511	36.88	10.85	26.32
	255	36.90	10.98	26.25
	127	36.84	10.87	26.39
	63	36.78	11.31	25.82
	31	36.67	11.40	25.64
	15	36.40	11.73	24.91
	13	35.09	11.56	23.80
	7	35.56	12.51	23.29
	3	34.21	15.36	19.40
	2	30.69	15.95	15.25
	1	25.32	18.66	11.51

The addition of MIS blocks extends the single-signal detection for all input amplitudes. The loss of 6 dB still exists when window function is used, but the dynamic range of the receiver improved by 22.37 dB to 54.17 dB using the hardware Hamming window implementation. SFDR is maintained for amplitude ranges of 511 to 7 before gradually waning off.

An additional extension to design 1 was simulated with variable truncation scheme for the outputs of the first 3 stages in the FFT. The resulting dynamic range performances for no window, ideal window, and hardware window implementations are 53.34, 40.19, and 38.6 dB. Comparing amongst the hardware window implementation results, VTS3 improved the receiver dynamic range to 38.6 dB from 31.8 dB, but it still pales in comparison with design 2, where the dynamic range is 54.17 dB.

7.3 FPGA Performance Verification

The VHDL representation of designs 1 and 2 are generated and programmed on the Virtex-4 FPGA. Single-signal and dual-tone signals tests are performed for both designs. Additionally, the impact of using 15 dB and 18 dB variable upper thresholds are also tested for dual-tone test. For both tests, the sampling frequency is 2.048 GHz with unsynchronized inputs (random phase).

The detected frequency values and their corresponding frequency spectrum magnitudes are tracked using Xilinx Chipscope Pro. These results are exported from Chipscope and evaluated in an automatic fashion using Matlab. The error margin set for the detected signal is 8 MHz. This allows frequency values situated exactly in between the signal bins to match to the closest frequency bin on each side.

7.3.1 Single-Tone Signal SFDR

The 10-bit Atmel ADC has a performance rating of 7.5 effective number of bits (ENOB) for a sampling frequency of 2 GHz. The ENOB rating is solely based on the Atmel ADC, it does not include the signal losses relative to the signal transfer from the ADC to the Xilinx FPGA LVDS connectors. No filters are used to remove the harmonic distortion from the signal generators because the harmonic distortion is outside the SFDR range of the implemented FFT.

The frequency is swept from 48 to 960 MHz at 8 MHz intervals for a total of 115 test cases per input amplitude. 512 continuous samples are recorded and analyzed for each test case, yielding a total of 58880 performance tests per input amplitude.

The signal generator setting required for generating the full scale values of the 10-bit ADC and LSB 8 bits (of the ADC) are at 3.6 and -9.3 dBm. Single-tone signal at 0 dBm is also tested for use as the strong signal for the dual-tone test. The amplitude of the signal is lowered continuously to investigate the sensitivity of the receiver design.

Both the full scale (FS) value and the weakest detectable signal (WS) amplitude (amp) with 100 % detection are used. The intent is to find the best possible SFDR of the receiver using the dynamic kernel function and variable truncation scheme No thresholds are set for the following single-signal analysis. The averaged performance results for designs 1 and 2 are tabulated in Tables 7.3 and 7.4.

The SFDR performance of design one using VTS seems to be higher for the strong signals utilizing the same order of bits (MSB 8 bits, LSB 8 bits of the ADC). Using MIS to process the input data, the sensitivity of the receiver is increased by 7 dB with the ability to 100% single-signal detection with amplitude of -40 dBm.

The MIS design is further tested with dual-thresholding, the signal can be lowered to -42 dBm, but with a lower percentage of detection. All the detected signals are still correct at this point with no false alarm. Pushing the input signal further to -46 dBm, the detection of the signal starts to reach 0% with some detected signals that are off by 1 frequency bin (16 MHz).

Table 7.3 FPGA Single-Signal Performance (Design 1 - VTS)

Amplitude (dBm)	Magnitude (dB)	Noise Spur (dB)	SFDR (dB)
3.6	39.67	12.93	26.70
0.0	36.85	12.54	24.38
-9.3	38.96	13.33	25.62
-33	18.64	11.69	7.21

Table 7.4 FPGA Single-Signal Performance (Design 2 - MIS)

Amplitude (dBm)	Magnitude (dB)	Noise Spur (dB)	SFDR (dB)
3.6	35.48	11.58	24.01
0.0	30.74	11.09	19.75
-9.3	32.95	12.07	20.90
-40	18.87	12.64	6.25

In comparison with the past 128-point FFT designs using 6-bit dynamic kernel [57] and 32 fixed-point kernels [58], the past implementations utilizes the conventional design techniques, where the LSB bits are constantly truncated away. The lowest detectable signals for these FPGA designs are around -18 to -22 dBm. Utilizing either VTS or MIS, sensitivity improvements of at least 11 and 18 dB are achieved. The previous 6-bit kernel implementation is different from the one used in this research work. The previous 6-bit kernel function butterfly truncates the intermediate partial fractions away which resulted in further approximation of the kernel.

Using the same analysis developed in Chapter 4 for Design 1, additions are made to Tables 4.3 and 4.4, where variable truncation scheme is applied after stage one only while the outputs after stages two to seven are constantly truncated, which results in Tables 7.5 and 7.6.

Table 7.5 Ideal Versus Dynamic Kernel Function SFDR Difference

Wordlength Consideration	No Window		Hamming Window	
	6-bits	10-bits	6-bits	10-bits
Continuous Growth	6.65 dB	3.65 dB	12 dB	7.2 dB
Constant Truncation	30.9 dB	28.2 dB	28 dB	26.7 dB
VTS (All Stages)	6.5 dB	3.8 dB	12.1 dB	7.25 dB
VTS (First Stage Only)	25.8 dB	22.6 dB	24.1 dB	21.2 dB

Table 7.6 Maximum Expected SFDR Using Dynamic Kernel Function

Wordlength Consideration	No Window		Hamming Window	
	6-bits	10-bits	6-bits	10-bits
Continuous Growth	54.6 dB	57.6 dB	41.9 dB	46.7 dB
Constant Truncation	30.5 dB	33.0 dB	25.9 dB	27.2 dB
VTS (All Stages)	54.5 dB	57.5 dB	41.9 dB	46.7 dB
VTS (First Stage Only)	35.4 dB	38.8 dB	29.9 dB	32.8 dB

For a 6-bit dynamic kernel function FFT with Hamming window applied, the maximum expected theoretical SFDR is 29.9 dB, about 3.2 dB difference from the FPGA verification results shown in Table 7.3. In addition, from Table 7.6, an improvement of at least 4 dB is observed for the maximum expected theoretical SFDR with variable truncation scheme applied to the output of stage one only, when compared against a constantly truncated FFT. This shows a promising trend for SFDR improvement if VTS can be applied to the output of all FFT stages in a larger FPGA.

Assessed with the ENOB of the 10-bit Atmel ADC, the expected dynamic range performance is about (6 dB/bit)*(7.5 effective bits), or about 45 dB. The MIS design has an overall dynamic range of (3.6 + 40) = 43.6 dB, which is fairly close to the estimated performance.

7.3.2 Dual-Tone Signal Test

The second signal detection and false alarm rates are the main evaluation parameters used for the different upper threshold settings. The dual-tone signal is composed of a strong and weak signal. The strong signal is maintained at 0 dBm while the weak signal amplitude is varied from -12 to -20 dBm to test the receiver's response to different IDRs.

The frequency difference between the two signals is 80 MHz, or five frequency bins. For each given IDR, the dual-tone signal is exhaustively swept from (48, 128) to (880, 960) MHz at 8 MHz intervals. This corresponds to 105 cases tested per IDR and 512 contiguous trials are collected per case using Xilinx Chipscope Pro debugging cores. The parentheses correspond to the frequency values of the combined signals. The increment in half bin intervals provides performance results when both signals are situated exactly on and between FFT bins.

For each pair of input signals, the code first checks for the detected primary signal. Then, for the trials with correctly detected primary signals, the code proceed to identify the second signal. The error margin set for the detected signal is also 8 MHz. This allows frequency values situated exactly in between the signal bins to match to the closest frequency bin on each side. Finally, the second signal detection and false alarm rates are

computed. For the detected signals, the minimum and average magnitudes in the frequency spectrum are also recorded to assess the appropriateness of the upper threshold setting for future work.

Given a set IDR value, each frequency sweep yields 105 cases with 512 trials. This result in total of 53760 samples, where 27136 samples are mapped to the bin (On bin) and 26624 samples are mapped to the closest bin (Off bin). The probability of detection (Pd) for second signal versus IDR is plotted in Fig. 7.5.

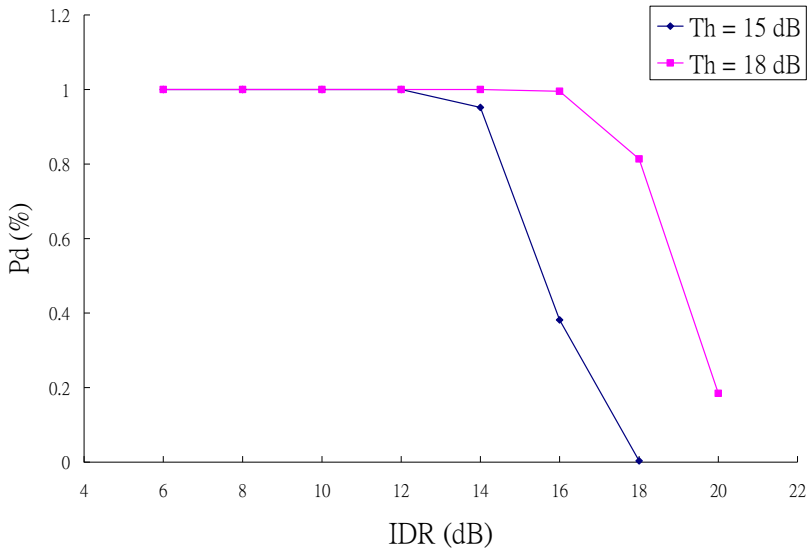

Fig. 7.5 Probability of Detection vs. Dual-Tone Signal IDR

The second signal detection rate is approximately 80 % when the IDR is close to the upper threshold value. However, the false alarm rates for the current experimental cases tested are all zero percent. In comparison with past non-parametric receiver designs [49-50], this is a significant improvement when the two-tone signal is 18 dB apart.

The second improvement is the detection of signals not represented by the frequency bins (Off bin). Typically, the magnitude of signals not centered on the frequency bins is about 3 dB lower when mapped to the closest FFT bin. Utilizing conventional digital receivers with one fixed threshold above the noise floor, these signals may not be detected at all. The exact percentage values are tabulated in Tables 7.7 and 7.8 for the second signal detection rate of the 53760 detected primary signals.

Table 7.7 Upper Threshold Setting (15 Decibels)

IDR	Second Signal Detection Rate (%)		
	Overall	*On Bin*	*Off Bin*
- 12	100.0000	100.0000	100.0000
- 14	95.1693	94.6924	95.4515
- 16	38.1864	31.2905	45.2148
- 18	0.3330	0.0037	0.6686

Table 7.8 Upper Threshold Setting (18 Decibels)

IDR	Second Signal Detection Rate (%)		
	Overall	*On Bin*	*Off Bin*
- 14	100.0000	100.0000	100.0000
- 16	99.5089	99.0492	99.9775
- 18	81.3261	79.1605	83.5374
- 20	18.4487	12.6253	24.3840

The preliminary but not conclusive results of dual threshold application show a promising alternative to adaptive thresholding for hardware constrained receiver designs. However, the utilization of dual thresholding may also be applied to current adaptively determined threshold receivers to further improve the second signal detection and false alarm rates.

8. CONCLUSION

8.1 Contributions

The main contributions of this research work are.

1. The development of hardware performance estimation models based on different number of bits used for dynamic kernel function. The model shows the relative trade-off between kernel bits to final FFT SFDR and phase performance. The algorithm may be further explored to yield the minimum number of bits needed to meet the intended FFT processor requirements. This will further cut down the amount of memory required to store the kernel functions. The dynamic kernel function can easily replace the twiddle factor representations in any existing FFT designs without making additional architectural modifications.
2. The utilization of variable truncation scheme with fixed inter-stage precision using fixed-point numbers provides an excellent alternative to floating or block floating-point representations to further reduce the hardware cost and design complexity associated with adders and multipliers. This is also a great method to test the input dynamic range of a given receiver when the bit precision is limited by the size of the prototyping FPGA.
3. Variable truncation scheme also maximizes the effective use of the fixed inter-stage bits without the need to continually expand the number of bits as data progresses to the output.

4. The design and implementation of a 2.048 GHz FFT processor with a throughput rate of 62.5 ns on a FPGA. No decimators were used in the design and all the sampled inputs were used. The 128-point FFT has an effective bandwidth of 912 MHz with 16-MHz channelization. The averaged single-signal SFDR while applying variable truncation scheme after stage one is 26 dB. Utilizing the same design setup, the two-tone signal IDR is close to 18 dB without any false alarms. The use of variable truncation scheme at the input improved the weakest detectable signal to -42 dBm versus -18 dBm when the scheme is not applied to any of the stages.

8.2 Future Work

Higher-Precision Data Format – The 128-point FFT implemented has a fixed-precision of 8-bits before and between the FFT stages, this was limited due to the amount of hardware resource available on the prototyping board. However, for future generations of FPGA or full-custom ASIC designs, the FFT may be designed without folding. This allows the use of variable truncation scheme after every single FFT stage while maintaining 8-bits for low hardware usage. The only drawback to this approach is that the single-signal SFDR and two-tone signal IDR does not increase with the dynamic range since the bit precision is still limited to 8-bits. New high-speed parallel architectures such as the systolic array [59] will need to be developed to maintain the high throughput rates without throwing away any sampled inputs.

Higher-Length FFT – The challenge of building higher-length FFTs with throughput rates greater than 62.5 is the ability of process the incoming data without losing any

sampled data. Similar to the higher-precision data format FFT, a new resource efficient architecture for wideband systems will need to be explored to deal with the handshaking issues.

Higher-Precision Dynamic Kernel Function – For radar and communications FFT processors, the magnitude and phase information in the frequency domain is important to decipher the modulated signal in addition to locating the target distance. The ability to do so in a noisy wideband environment without any priori information is difficult. Better thresholding are needed in addition to higher-precision dynamic kernel function. Careful considerations for both inter-stage and dynamic kernel function bit-precisions are required to minimize the hardware overhead. The model used for this research should be a good starting point to find the balance between these two parameters.

9. REFERENCES

[1] P. Clarke, EE Time: Intel enters billion-transistor processor era, Nov. 14, 2005.

[2] P. H. W. Leong, "Recent trends in FPGA architectures and applications," in *Proceedings of IEEE International Symposium on Electronic Design, Test, and Applications*, Jan. 2008, pp. 137-141.

[3] J. Marcum, "A statistical theory of target detection by pulsed radar," *IRE Trans. on Information Theory*, vol. 6, no. 2, pp. 59-267, Apr. 1960.

[4] E. O. Brigham. *The Fast Fourier Transform and Its Applications*. Englewood Cliffs, Prentice-Hall, NJ, 1988.

[5] R. N. Bracewell. *The Fourier Transform and its Applications*. Mcgraw-Hill, New York, NY, second edition, 1986.

[6] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-Time Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ, second edition, 1999.

[7] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*. Addison-Wesley, Reading, MA, 1987.

[8] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice-Hall, third edition, 1996.

[9] D. J. DeFatta, J. G. Lucas, and W. S. Hodgkiss. *Digital Signal Processing: A System Design Approach*. John Wiley & Sons, New York, NY, 1988.

[10] L. B. Jackson. *Digital Filters and Signal Processing*. Kluwer Academic, Boston, MA, 1986.

[11] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of the complex Fourier series," *Math. Of Computation*, vol. 19, no. 90, pp. 297-301, Apr. 1965.

[12] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the history of the fast Fourier transform." *IEEE ASSP Magazine*, vol. 1, no. 4, pp. 14-21, Oct. 1984.

[13] C. Runge, *In Zeit. fur Math. und Physik*, vol. 48, p. 443, 1903.

[14] C. Runge, *In Zeit. fur Math. und Physik*, vol. 53, p. 117, 1905.

[15] G. C. Danielson and C. Lanczos, "Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids," *J. Franklin Inst.*, vol. 233, no. 4 and no. 5, pp. 365-380 and 435-452, Apr. and May, 1942.

[16] R. C. Singleton, "A method for computing the fast Fourier transform with auxiliary memory and limited high-speed storage," *IEEE Trans Audio and Electroacoustics*, vol. 15, no. 2, pp. 91-98, Jun. 1967.

[17] R. C Singleton, "An algorithm for computing the mixed radix fast Fourier transform," *IEEE Trans. Audio and Electroacoustics*, vol. 17, no. 2, pp. 93-103, Jun. 1969.

[18] P. Duhamel and H. Hollmann, "Split-radix FFT algorithms," *Electronics Letters*, vol. 20, no. 1, pp. 14-16, Jan. 1984.

[19] P. Duhamel, "Implementation of split-radix FFT algorithms for complex, real, and real-symmetric data," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 34, no. 2, pp. 285-295, Apr. 1986.

[20] P. N. Swarztrauber, "Symmetric FFTs," *Mathematics of Computation*, vol. 47, pp. 323-346, Jul. 1986.

[21] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *American Math. Monthly*, vol. 65, no. 1, pp. 34-35, Jan. 1958.

[22] L. I. Bluestein, "A linear filtering approach to the computation of discrete Fourier transform," *IEEE Trans. Audio and Electroacoustics*, vol. 18, no. 4, pp. 451-455, Dec. 1970.

[23] L. R. Rabiner, R. W. Schafer, and C. M. Rader, "The chirp z-transform algorithm," *IEEE Trans. Audio and Electroacoustics*, vol. 17, no. 2, pp. 86-92, Jun. 1969.

[24] C. M. Rader, "Discrete Fourier transform when the number of data samples is prime," *Proc. IEEE*, vol. 56, pp. 1107-1108, Jun. 1968.

[25] I. J. Good, "The relationship between two fast Fourier transforms," *IEEE Trans. Computers*, vol. C-20, no. 3, pp. 310-317, Mar. 1971.

[26] S. Winograd, "On computing the discrete Fourier transform," *Math. Comp.*, vol. 32, no. 141, pp. 175-199, Jan. 1978.

[27] C. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 25, no. 3, pp. 239-242, Jun. 1977.

[28] C. S. Burrus and T. W. Parks. *DFT/FFT and Convolution Algorithms and Implementation*. Wiley, New York, 1985.

[29] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, "A 1-GS/s FFT/IFFT processor for UWB applications," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 8, pp. 1726-1735, Aug. 2008.

[30] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp 310-319, Jan. 2005.

[31] J.-C. Kuo, C.-H. Wen, C.-H. Lin, and A.-Y. Wu, "VLSI design of a variable-length FFT/IFFT processor for OFDM-based communication systems," *EURASIP J. Applied Signal Processing*, vol. 2003, no. 13, pp. 1306-1316, Jul. 2003.

[32] Texas Instruments, *Application Report SPRA948*, pp. 1-12.

[33] Y. Chen, Y.-C. Tsao, Y.-W. Lin, and C.-Y. Lee, "An indexed-scaling pipelined FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits and Systems-II, Express Briefs*, vol. 55, no. 2, pp. 146-150, Feb. 2008.

[34] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 complex point FFT," *IEEE J. Solid-State Circuits*, vol. 30, no. 3, pp. 300-305, Mar. 1995.

[35] J.-R. Choi, S.-B. Park, D.-S. Han, and S.-H. Park, "A 2048 complex point FFT architecture for digital audio broadcasting system," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2000, vol. 5, pp. 693-696.

[36] T. Lenart and V. Owall, "A 2048 complex point FFT processor using a novel data scaling approach," in *Proc. IEEE Int. Symp. Circuits Syst*., May 2003, vol. 4, pp. 45-48.

[37] T. Lenart and V. Owall, "Architecture for dynamic data scaling in 2/4/8K pipeline FFT cores," *IEEE Trans. VLSI Syst*., vol. 14, no. 11, pp. 1286-1290, Nov. 2006.

[38] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Trans. Circuits and Systems*, vol. 39, no. 5, pp. 312-316, May 1992.

[39] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Trans. on Circuits and Systems-I, Regular Papers*, vol. 52, no. 5, pp. 911-919, May 2005.

[40] T. Chen, G. Sunada, and J. Jin, "COBRA: A 100-MOPS single-chip programmable and expandable FFT," *IEEE Trans. VLSI Syst.*, vol. 7, no. 2, pp. 174-182, Jun. 1999.

[41] B. M. Baas, " A low-power, high-performance, 1024-point FFT processor," *IEEE J. Solid-State Circuits*, vol. 34, no. 3, pp. 380-387, Mar. 1999.

[42] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, "A dynamic scaling FFT processor for DVB-T Applications," *IEEE J. Solid-State Circuits*, vol. 39, no. 11, Nov. 2004.

[43] M. Hasan, T. Arslan, and J. S. Thompson, "A novel coefficient ordering based low power pipelined radix-4 FFT processor for wireless LAN applications," *IEEE Trans on Consumer Electronics*, vol. 49, no. 1, Feb. 2003.

[44] K. K. Parhi, "Approaches to low-power implementations of DSP systems," *IEEE Trans. Circuits and Systems-I, Fundamental Theory and Appliations*, vol. 48, no. 10, pp. 1214-1224, Oct. 2001.

[45] S. C. Chan P. M. Yiu, "An efficient multiplierless approximation of the fast Fourier transform using sum-of-powers-of-two (SOPOT) coefficients," *IEEE Signal Processing Letters*, vol. 9, no. 10, pp. 322-325, Oct. 2002.

[46] Y. Jung, H. Yoon, and J. Kim, "New efficient FFT algorithm and pipeline implementation results for OFDM/DMT applications," *IEEE Trans. on Consumer Electronics*, vol. 49, no. 1, pp. 14-20, Feb. 2003.

[47] J. B. Y. Tsui and J. P. Stephens, "Digital Microwave Receiver Technology," *IEEE Trans. on Microwave Theory and Techniques*, vol. 50, no. 3, pp. 699-705, Mar. 2002.

[48] D. Pok, C.-I. H. Chen, J. Schamus, C. Montgomery, and J. B. Y. Tsui, "Chip Design for Monobit Receiver," *IEEE Trans. on Microwave Theory and Techniques*, vol. 45, no. 12, pp. 2283-2295, Dec. 1997.

[49] C.-I. H. Chen, K. George, W. McCormick, J. B. Y. Tsui, S. L. Hary, and K. M. Graves, "Design and Performance Evaluation of a 2.5-GSPS Digital Receiver," *IEEE Trans. on Instrument. and Measurement*, vol. 54, No. 3, pp. 1089-1099, Jun. 2005.

[50] K. George, C.-I. H. Chen, and J. B. Y. Tsui, "Extension of Two-Signal Spurious-Free Dynamic Range of Wideband Digital Receivers Using Kaiser Window and Compensation Method," *IEEE Trans. on Microwave Theory and Techniques*, vol. 55, no. 4, pp. 788-794, Apr. 2007.

[51] J. M. Emmert, J. A. Cheatham, B. Jagannathan, and S. Umarani, "An FFT Approximation Technique Suitable for On-Chip Generation and Analysis of Sinusoidal Signals," in *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov. 2003, pp. 361-368.

[52] J. B. Y. Tsui. *Digital Techniques for Wideband Receivers.* Second Ed., Norwood, MA: Artech House, 2001.

[53] Y.-H. G. Lee and C.-I. H. Chen, "Dynamic kernel fast Fourier transform with variable truncation scheme for wideband coarse frequency detection," *IEEE Trans. Instrumentation and Measurement*, vol. 58, no. 5, pp. 1555-1562, May 2009.

[54] P. Pirsch. *Architectures for Digital Signal Processing.* New York, Wiley, 1998.

[55] Agilent Technologies, *Agilent N9310A RF signal generator – technical overview*, pp. 9.

[56] Agilent Technologies, *Agilent 8648 A/B/C/D signal generators – datasheet*, pp. 2.

[57] Vivek Sarathy. "High spurious-free dynamic range digital wideband receiver for multiple signal detection and tracking." M.S. thesis, Wright State University, U.S.A., 2007.

[58] Ryan Bone. "FPGA design of a hardware efficient pipelined FFT processor." M.S. thesis, Wright State University, U.S.A., 2008.

[59] P. A. Jackson, C. P. Chan, J. E. Scalera, C. M. Rader, and M. Vai, "A systolic FFT architecture for real-time FPGA systems," in *Proceedings of 8th Annual High Performance Embedded Computing Workshop*, Lexington, MA, Sep. 2004.